Approximating Expressive Queries on Graph-modeled Data: the GEX Approach[☆]

Federica Mandreoli[a], Riccardo Martoglia[a], Wilma Penzo[b,*]

[a] *FIM, University of Modena and Reggio Emilia, via Campi, 213/b, I-41125 Modena, Italy*
[b] *DISI, University of Bologna, viale Risorgimento, 2, I-40136 Bologna, Italy*

Abstract

We present the GEX (Graph-eXplorer) approach for the *approximate matching of complex queries* on graph-modeled data. GEX generalizes existing approaches and provides for a highly expressive graph-based query language that supports queries ranging from keyword-based to structured ones. The GEX query answering model gracefully blends label approximation with structural relaxation, under the primary objective of delivering *meaningfully approximated* results only. GEX implements ad-hoc data structures that are exploited by a top-k retrieval algorithm which enhances the approximate matching of complex queries. An extensive experimental evaluation on real world datasets demonstrates the efficiency of the GEX query answering.

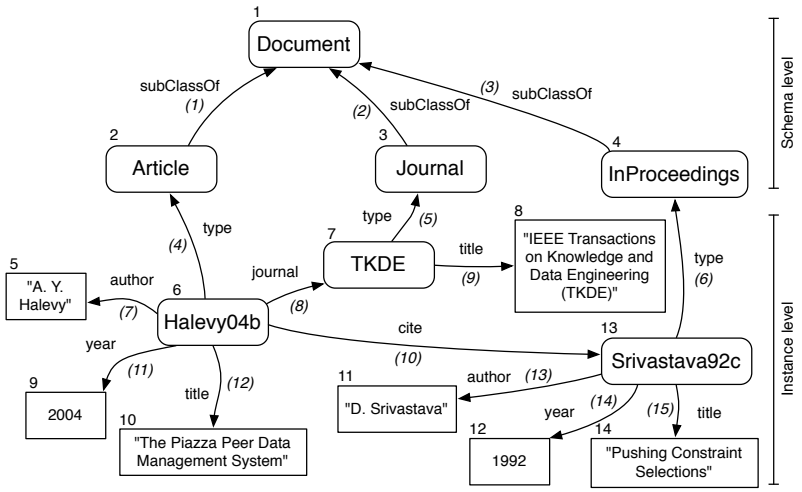
Keywords: Graph-modeled data, Complex queries, Approximate graph query answering, Semantic Relatedness, Graph indexing, Top-k query answering

1. Motivation

The graph-based modeling paradigm has recently regained much popularity thanks to the dramatic increase of information sources that find in graphs a natural way of modeling data. Among these, we can mention data publicly available on the Web, social networks, personal health records, biological and chemical databases, etc., where datasets are usually modeled through complex semantic data graphs characterized by domain largeness and data heterogeneity. The available structured query models (e.g. SPARQL for RDF datasets) do not comply to these distinctive features, because they require users to be knowledgable about the queried structure in order to precisely specify both the locations of the entities they are searching for, and the relationships among them [36], de facto making exact querying of this kind of data impractical.

[☆] A preliminary version of this work was presented in [25].

[*] Corresponding author. Tel: +39 051 20 9 3560
Email addresses: federica.mandreoli@unimo.it (Federica Mandreoli), riccardo.martoglia@unimo.it (Riccardo Martoglia), wilma.penzo@unibo.it (Wilma Penzo)



Query 1: The title of the papers authored by someone whose name sounds like 'Sivastava' published before 2005.
Query 2: How the 'Piazza' paper published in the TKDE journal relates to other documents.

Figure 1: Reference graph and two query samples

To overcome these limitations, several research efforts are currently being spent to develop effective retrieval techniques which allow users to easily query graph-based data and to get useful results by means of *flexible query mechanisms* [8, 19, 29, 31, 34].

One way to achieve flexibility in query formulation is to adopt a keyword-based query model [15, 18, 22, 27]. This solution has the merit of eliminating structures in the query, thus lightening the user from the burden of knowing the relationships occurring between the data. Furthermore, it easily applies to heterogeneous scenarios where multiple schemas coexist. On the other hand, a keyword-based approach suffers from an inherently limited capability in the query semantics that it can express. We are all familiar with how difficult it is to translate a complex request in a set of keywords, and most of all to get precise answers from this means.

For instance, a sample of graph-modeled data and two possible user information needs that can be answered on it (Query 1 and Query 2) are shown in Fig. 1. The graph represents an excerpt from the RDF version of the DBLP scientific bibliography database[1] showing a citing relationship between two publications. By following a simple keyword search query model [15, 22] a user could translate

[1] publicly available at `http://sw.deri.org`. Because of its extensive and dense network of relations, DBLP is a good dataset for formulating complex graph-based queries and is one of the most exploited resources in the literature for graph search testing purposes [15, 30, 23].

Query 1 into the set of keywords `Q1={title, paper, author, Srivastava, 2005}`.[2] Answers to this query are a subtree [15] or a subgraph [18, 22, 27] of the data graph connecting all keywords. The main drawback of this kind of querying approach is that answers would actually contain the queried keywords but they will be related by just *topological* connections, so completely disregarding the semantics underlying the relationships between them. This means that, for instance, also papers that cite papers by `Srivastava`, as well as papers having `2005` as page number, would appear in the result, although they are clearly not relevant for the query.

Querying complex semantic data graphs where entities are related through named relationships motivates the need for supporting querying capabilities that go beyond keyword-based search when searching for knowledge [14, 19]. In this case, users usually have focused and often complex information needs, and they should be enabled to include varying degrees of structure in their queries, so that they can better specify a mix of vague, precise and implicit requirements according to the partial knowledge of the schema they may have.

Some proposals in the literature have worked in this direction and present graph-based approaches [8, 10, 19, 29, 34, 38, 40], among which only NAGA [19] and the work in [10] allow for the specification of named relationships in the query graphs. However, these works approximate query edges by obeying to mere topological conditions, i.e. edges are approximated by paths. This answering model often returns pieces of information that are not actually related to each other. For instance, neither of them is able to properly answer Query 2 shown in Fig. 1, where the kind of connection between the TKDE 'Piazza' paper and other documents is not specified. Results from the approaches above would be about documents connected to the given paper along *any* topological path, thus suffering from the same drawbacks discussed about keyword-based approaches. This has crucial implications, since in many domains it is of fundamental importance to get only *meaningful* answers, i.e., answers made up of data related in a significant way.

The issue of retrieving meaningful answers has been initially investigated in the area of XML data search [6, 36]. However these proposals are limited to support keyword search [6] and are targeted for XML data [24, 36], thus they do not fit for the purpose of answering edge-labeled graph queries. The work in [10] goes a step beyond by constraining graph results on (sub)path lengths. Essentially it founds on the assumption that entities sufficiently close together are meaningfully related. However, this work too relies on topological approximations.

Also, currently there is no query model supporting Query 1, i.e. a query containing two predicates that specify constraints on the data (document's author name is required to sound like 'Sivastava' and publication year has to be prior to 2005). Indeed, up to now most of the efforts have been spent on the

[2] For the sake of simplicity, for the time being, we assume to disregard the 'Sivastava' misspelling.

problem of approximating the structure of a query [3, 10, 29, 34, 40], often giving little attention to support vagueness on the used vocabulary. Rather, it has been proved that label ambiguity is a very frequent issue, because people hardly choose the same term for a single well-known object [12]. For this purpose, some works [7, 42] adopt term expansion techniques, and follow a query relaxation approach. The framework in [44] supports node label substring operations on RDF graphs. However, neither of these works do explicitly deal with vagueness on query edge labels.

In this paper we present the GEX (Graph-eXplorer) approach to support the *approximate matching of complex queries* on graph-modeled data. The major contributions of GEX are:

- it offers a framework that *generalizes* the existing approaches and allows for querying any datasets which conform to a graph representation (e.g. relational databases, XML data, RDF repositories);
- it provides for a highly expressive graph-based query language to formulate queries ranging from keyword-based to complex ones, including (Section 2):
  1. the specification of relationships between data;
  2. the expression of vague and missing information both on data and on their relationships;
  3. the specification of constraints on the data;
- it introduces the notion of *Semantic Relatedness*, which identifies pairs of nodes *meaningfully related* in a data graph, i.e. nodes related to each other in a significant way (Section 3.1);
- it supports the approximate matching of vague, precise, and implicit query requirements by gracefully blending label approximation with structural relaxation, under the primary objective of delivering *meaningfully approximated* results only (Section 3.2). To this purpose, GEX relies on the introduced notion of Semantic Relatedness;
- it introduces a novel indexing scheme that implements the notion of Semantic Relatedness to efficiently support the approximate matching on graph-modeled data (Section 4);
- it implements an efficient top-k retrieval algorithm which further accelerates query answering by taking advantage of two interesting properties exhibited by the GEX ranking model (Section 5);
- it demonstrates the efficiency of the query answering approach through a comprehensive experimental evaluation on different real world datasets (Section 6).

Finally, Section 7 discusses related work, while Section 8 concludes the paper and outlines future research directions.

This paper extends and improves our previous work [25] in almost every respect. A large part of the improvement effort has been focused on the design of

the novel indexing scheme and on the analysis of different implementation strategies for the involved data structures (Section 4). In particular, we practically show that the new scheme accelerates complex query processing tasks (Section 6) and provides crucial advantages both in terms of querying performance (one order of magnitude faster execution times, on average) and storage requirements (indexing space reduced to as little as 4% of the original requirements). We also provide new effectiveness and efficiency comparisons, highlighting the benefits of the GeX approach w.r.t. the most relevant state of the art proposals, and evaluating the impact of the different implementation strategies.

2. Preliminaries: the GeX Data and Query Models

In this Section we briefly present the GeX models for representing and querying graph data. For more detailed descriptions, the interested reader can refer to [25].

2.1. The Data Model

In order to ensure a uniform access to graphs referring to different data models, GeX adopts a *general model* which covers RDF and OWL data as well as most graph-based data models recently introduced in the literature for various purposes (e.g. [7, 14, 19, 22]).

The model represents data universally as a connected graph allowing parallel edges (also known as multigraphs) and nodes and edges possibly labelled. It is worth noting that parallel edges are necessary to model RDF and OWL based data as well as dataspaces [11], although not all general purpose models support them (e.g. [29]). In the same way, unlabeled edges allow for modeling those kinds of data graphs where both labeled and unlabeled edges coexist, such as XML where parent-child relationships, represented as unlabeled edges, need to be distinguished from cross-reference links, usually labeled through IDs/IDREFs.

In the following, we denote as L the set of all possible labels partitioned in literal values L_L (e.g. `2004`, `"A. Y. Halevy"`) and concept labels L_C (e.g. `Article`, `author`, `cite`). Besides, the empty label ϵ is used to denote unlabeled edges.

Definition 1 (Data Graph). *Data is represented as a connected directed labeled multi-graph* $\mathcal{G} = (N, E, L_N, L_E)$ *where*

- N *is a set of nodes,*
- $E \subseteq N \times N$ *is a multiset of directed edges,*
- $L_N \subseteq L$ *and* $L_E \subseteq L_C \cup \{\epsilon\}$ *are sets of node and edge labels, respectively. Each node* $n \in N$ *is assigned a label* $\lambda(n) \in L_N$ *and each edge* $e \in E$ *is assigned a label* $\lambda(e) \in L_E$*.*

Nodes having labels in L_L *are called* value nodes *whereas nodes with labels in* L_C *are called* entity nodes*.*

Example 2.1. *Let us consider again Fig. 1 which will be used as reference example. It contains the RDF classes* `Article`, `Journal`, `InProceedings`, *and* `Document`, *the RDF schema properties* `subClassOf` *and* `type`, *three class instances,* `Halevy04b`, `Srivastava92c`, *and* `TKDE`, *many RDF properties, such as* `author` *and* `year`, *and literals, such as* `"A. Y. Halevy"`. *For ease of presentation, property specifications are not provided at schema level.*

The above graph as well as any other RDF graph translates into our data model in the following way: RDF classes, class instances, and any other resource are modeled as entity nodes, depicted in Fig. 1 as rounded rectangles, whereas RDF literals are modeled as value nodes, depicted in Fig. 1 as rectangles.

2.2. The Query Model

In order to query the data graph, GEX provides an *expressive graph-based query language* which goes beyond the keyword-based approach without necessarily requiring to precisely use the graph vocabulary and structure in query formulation. As a matter of fact, a single query graph can both express focused information requests and contain different pieces of missing information as well as "vague" connections.

To this extent, the GEX query language puts at the user disposal several degrees of flexibility. It supports undirected edges and unbound nodes and edges through variables, thus allowing users to leave nodes and edges (i.e. relationships) between them (partially) unspecified according to the knowledge they may have about the data schema. Then, both unbound nodes and edges can be constrained by conditions. In this way, node connection types range from the fully specified connection, i.e. a labeled and directed edge between two labeled nodes, to the fully unbound connection, i.e. an unbound and undirected edge connecting two unbound nodes.

Definition 2 (Query Graph). *A query is a tuple* $q = (N^q, E^q, L^q_N, L^q_E, V, C)$ *where* $E^q = E^q_d \cup E^q_u$ *and*

- (N^q, E^q_d) *is a directed multi-graph,*
- (N^q, E^q_u) *is an undirected multi-graph,*
- $L^q_E \subseteq L_C$ *and each* $e \in E^q$ *is assigned a label* $\lambda(e) \in L^q_E \cup V$,
- $L^q_N \subseteq L_C$ *and each* $n \in N^q$ *is assigned a label* $\lambda(n) \in L^q_N \cup V$,
- (N^q, E^q, L^q_N, L^q_E) *is a connected labeled multi-graph,*
- V *is a set of variables,*
- C *is a set of conditions on* V. *Each condition in* C *has the form* $var\langle op\rangle v$ *where* $var \in V$, $\langle op\rangle$ *is an operator, and* $v \in L_L$ *is a value. Possible operators are the relational ones, i.e.,* $=, <, \leq, >, \geq$, *and the two operators* $\ni$ *and* $\sim$. *The semantics of the* $\ni$ *operator is the usual term containment in a text, whereas the* $\sim$ *operator expresses similarity between literals.*

Nodes and edges having labels in L^q_N *are called* labeled nodes *and* labeled edges *while nodes and edges having labels in* V *are called* unbound nodes *and* unbound edges*, respectively.*

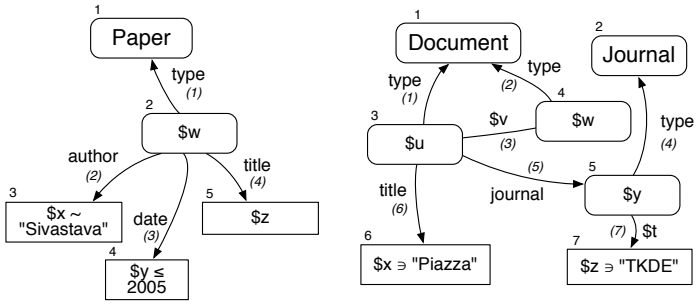


Figure 2: Query 1 (left) and Query 2 (right)

Example 2.2. *Fig. 2 shows the* GEX *queries corresponding to the two information needs shown in Fig. 1. Note that Query 2 contains a fully unbound connection, i.e. edge e_3 (for ease of reference, nodes and edges in data or query graphs are univocally identified by integer numbers i and will be referenced as n_i and e_i, respectively).*

None of them finds an exact match on the reference graph because they contain imprecise specifications. For instance, node `Paper` *and edge* `date` *of Query 1 find no exact correspondence in the data vocabulary while no edge labeled* `type` *with tail labeled* `Document` *as depicted in Query 2 exists in the data structure.*

In the following we will show how GEX deals with these kinds of requests in an effective and efficient fashion.

3. The GEX approximate query answering model

The GEX framework founds on an *approximate subgraph matching* approach which matches the query constraints by effectively dealing with the possibly contained ambiguities, both in the labels and in the connections between nodes. In this way, GEX supports not only imprecise query specifications but also heterogeneous datasets where exact subgraph matching on the different coexisting structures would often fail to produce useful results.

Approximate subgraph matching is performed in a two-fold fashion. As we allow label ambiguities, where the exact label names are unknown, the first kind of approximation we consider are node/edge label mismatches. To this end, the degree of mismatch between concept labels is quantified by means of the function $d_L : (L_C \cup V) \times L_C \rightarrow [0, 1]$. It is a scoring function which returns a value ranging from exact match (0) to total mismatch (1). Obviously, for all labels $l \in L_C$, $d_L(\$x, l) = 0$, $d_L(l, l) = 0$ and $d_L(\epsilon, l) = 1$. As far as its definition is concerned, we let the users customize the label matching method that best suits the application needs. In [25] we proposed an adaptation of the Leacock-Chodorow [21] distance, which relies on the WordNet thesaurus[3] to compare

[3] http://wordnet.princeton.edu

the hypernymy hierarchies of two disambiguated labels.

Moreover, as users are not required to know the graph topology, we also support structural approximations. Regarding this last point, GEX pursues the objective of delivering *meaningfully approximated* results only. To this end, GEX approximatively matches adjacencies only with those paths in the data graph that meaningfully relate node pairs, i.e. related in a significant way. For instance, referring to Fig. 1, note that `2004` is not the year of publication of the paper `Srivastava92c` and thus none of the paths connecting node n_{13} with node n_9 should be considered when approximating the `date` edge of Query 1. On the other hand, `Srivastava92c` is an instance of the `Document` class and the path connecting n_{13} to n_1 can be used to approximate the `type` edge of Query 2.

An approximate subgraph matching algorithm often returns a large number of query results and an effective ranking mechanism is an essential means to sort the matches based on their similarities to the query. To this end, GEX delivers the *top-k answers* to a query by implementing a general definition of scoring function which relies on both graph structures and content to promote the most compact semantically related results.

3.1. On the Meaningfulness of Structural Approximations

When querying graph-based datasets, different matching techniques can be applied. On the one hand, exact graph matching maps each node adjacency specified in the query graph to exactly one data graph edge. On the other hand, one viable solution to support structural approximation is the topological approach which relaxes adjacency constraints by allowing node/edge insertions in the data graph. Moreover, since in a connected data graph all nodes are pairwise connected by at least one path, in order to cope with the possibly large amount of approximations, some syntactic properties on the involved labels and/or on the missed exact matches should be checked. This is the approach adopted, for instance, in [10, 19, 29].

However, it should be noted that the fact that nodes are topologically connected does not necessarily imply that they are meaningfully related, and humans can often determine whether they are or not by looking at the graph. Essentially, we are concerned with graph-modeled data where nodes represent entities in the real world and edges relationships among them. Therefore, we can leverage the graph topology semantics to decide whether any pair of nodes is meaningfully related and, in case, to annotate such a linkage with a label.

To our knowledge, the requirement that the elements satisfying a query must be meaningfully related has been first introduced in [6] for labeled keyword search over XML documents. In that context, meaningfully related nodes are used for conjunctive query answering. In our context, instead, they represent the terminal points to be taken into account when approximating node adjacency.

Generally speaking, it is difficult to determine when a set of elements is meaningfully related. Therefore, we assume that there is a given relationship that determines when two nodes are related. This leads to the notion of *Semantic Relatedness* on a data graph $\mathcal{G}$ as that relation that only contains pairs of

nodes in $\mathcal{G}$ which are meaningfully related. We also give one natural example of such a relation for an RDF-like data model (see Appendix A) which was used in our experiments. However, it is possible to use a different relation, with no impact on system efficiency.

Before presenting the concept of Semantic Relatedness, we introduce the notion of path in a data graph $\mathcal{G}$.

Definition 3 (Path). *Given a graph $\mathcal{G}$, a* path *in $\mathcal{G}$ is a sequence $p = \langle e_1, \ldots, e_m \rangle$ of consecutive edges between nodes in $\mathcal{G}$. Given a path $p = \langle e_1, \ldots, e_m \rangle$ the* length *of p is the number m of traversed edges in p.*

For ease of reference, the singleton $\langle e \rangle$ will be simplified as e.

A sample of path of length 3 in the graph depicted in Fig. 1 is $\langle e_4, e_1, e_2 \rangle$. Note that the involved edges do not have to share the same direction.

Definition 4 (Semantic Relatedness (SR)). *Given a data graph $\mathcal{G} = (N, E, L_N, L_E)$, the Semantic Relatedness relation SR over $\mathcal{G}$ is a bag that is assumed to contain node pairs $(n, n') \in N \times N$ that are meaningfully related. Each $e = (n, n') \in SR$ is assigned a label $\lambda(e) \in L_C$ and a path $p(e)$ in $\mathcal{G}$ connecting n with n'.*

SR must satisfy two properties:

- *the* graph-containment *property: $E \subseteq SR$;*
- *the* path-decomposition *property: for each pair $e = (n, n') \in SR$ such that $p(e) = \langle e_1, \ldots, e_m \rangle$ and $m > 1$, there is an edge $e_j = (n_{j-1}, n_j) \in p(e)$ such that $e' = (n, n_j) \in SR$ and $e'' = (n_j, n') \in SR$; and $p(e)$ is the concatenation of $p(e')$ with $p(e'')$.*

The next definition about SR introduces a function that quantifies the cost of approximating each instance of SR on a data graph G. Such a quantification allows the ranking of the paths in SR.

Definition 5 (Approximation cost function). *Given a data graph $\mathcal{G} = (N, E, L_N, L_E)$ and a semantic relatedness relation SR over $\mathcal{G}$, an approximation cost function c for SR is a monotonically increasing function defined over the paths of $\mathcal{G}$ such that $c(p(e)) = 0$, for each $e \in SR \cap E$. Given an edge $e \in SR$, $c(p(e))$ will be denoted as $c(e)$ for brevity.*

In other words, each instance $e \in SR$ is a "virtual" edge which $p(e)$ approximates in $\mathcal{G}$, and $c(e)$ represents the cost of approximating e with $p(e)$. $\lambda(e)$ is a concept label for the relationship connecting the nodes in e. The graph containment property straightforwardly follows from above, since no structural approximation is required for each edge $e \in E$. Instead, the path-decomposition property follows from the fact that if two nodes are meaningfully related by means of other nodes of the graph, then, intuitively, they are also meaningfully related with such nodes.

SR is a bag and thus can contain multiple occurrences that share the same edge vertices and label and only differ in the path. Essentially they represent

the same relationship. In order to overcome such a redundancy, we propose to adopt the *distinct-node set semantics*. In particular, we introduce a reduced version of SR where only the "less expensive" node pairs are retained.

Definition 6 (Reduced SR). *Let $\mathcal{G}$ be a data graph and SR be a semantic relatedness relation over $\mathcal{G}$. The reduced version of SR, denoted as $\overline{SR}$, contains each node pair $e = (n, n') \in SR$ such that for all $e' = (n, n') \in SR$ such that $d_L(\lambda(e), \lambda(e')) = 0$ then $c(e) < c(e')$.*

In Def. 6 the condition expressed through d_L means that $\lambda(e)$ and $\lambda(e')$ should denote the same relationship.

We have several reasons for adopting a distinct-node set semantics. Firstly, it is an intuitive and clean semantics that preserves the semantics of query answering since the set of answers built over any semantic relatedness relation is the same as the one obtained from its reduced version. Secondly, it favors the less expensive structural approximations which similarly correspond to the most compact answers [15, 19] since the approximation cost function is monotonic. On the other hand, all answers sharing the same edge vertices and labels overlap and each answer carries very little additional information from the rest. Thus, by adopting a distinct-node set semantics we avoid to overwhelm users with quasi-redundant results. The last reason is more technical: it reduces the amount of data required for query answering, thus impacting on query processing efficiency.

3.2. The Query Answering Model

Both label and structural approximation are used to introduce the notion of query answer. To this end, we define two assignment functions f and g that deal with node mismatches and adjacency misses in query answers, respectively. In particular, f allows for node label approximations while g allows for both edge label approximations and adjacency approximations.

Definition 7 (Query answer). *Let $\mathcal{G}$ be a data graph, $\overline{SR}$ be a reduced semantic relatedness relation over $\mathcal{G}$, and $q = (N^q, E^q, L^q_N, L^q_E, V, C)$ be a query. An $\overline{SR}$-answer to q is an* approximate embedding $\mathcal{E}^{\overline{SR}} = (f, g)$ *where*

- *f is an injective* node-assignment function $f : N^q \to N$ *satisfying the following constraints:*
  - *for every labeled node n, $f(n)$ is an entity node and $d_L(\lambda(n), \lambda(f(n))) < 1$;*
  - *for every condition $c = \lambda(n)\langle op \rangle v$ in C, $f(n)$ is a value node and $\lambda(f(n))\langle op \rangle v$ holds with a certain grade $s(c)$ which is called the* score *of c. In particular, relational operators have a Boolean semantics and thus $s(c)$ must be 1 whereas operators $\ni$ and $\sim$ return values $s(c) \in [0, 1]$ and it must be $s(c) > 0$;*
- *g is an injective* edge-assignment function $g : E^q \to \overline{SR}$ *where $g(e)$ satisfies the following constraints for each $e = (n, n') \in E^q$:*

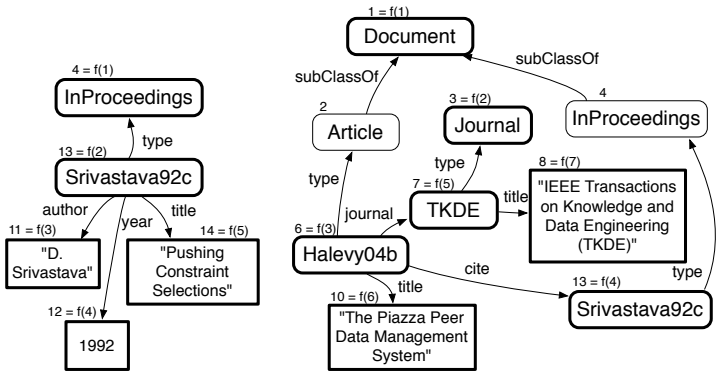


Figure 3: Embeddings for Query 1 (left) and for Query 2 (right)

- $d_L(\lambda(e), \lambda(g(e))) < 1$;
- $g(e) = (f(n), f(n'))$, *if* e *is a directed edge, i.e.* $e \in E_d^q$, $g(e) = (f(n), f(n'))$ *or* $g(e) = (f(n'), f(n))$, *otherwise.*

Each embedding $\mathcal{E}^{\overline{SR}}$ of a query graph q defines a subgraph of $\mathcal{G}$ which consists of the set of nodes in $f(N^q)$ connected through the paths in $p(g(E^q))$. To this end, it is worth noting that any exact embedding is an approximate embedding.

Example 3.1. *The subgraphs depicted in Fig. 3 show one plausible approximate embedding on the reference data graph of Fig. 1 for each of the two query samples.*

For each embedding, the data nodes in the range of f are depicted in bold line and the query node image is shown on the left upper corner of each data node rectangle. For instance, $4 = f(1)$ means that data node n_4 is assigned to query node n_1.

As to edge-assignment, it founds on the set of $\overline{SR}$ edges that can be derived by applying the rules shown in Appendix A to the reference example. In particular, the edge-assignment function of embedding on the left side is defined as

e	$g(e)$	$\lambda(g(e))$	$p(g(e))$
e_1	(n_{13}, n_4)	*type*	e_6
e_2	(n_{13}, n_{11})	*author*	e_{13}
e_3	(n_{13}, n_{12})	*year*	e_{14}
e_4	(n_{13}, n_{14})	*title*	e_{15}

while for the embedding on the right side

e	$g(e)$	$\lambda(g(e))$	$p(g(e))$
e_1	(n_6, n_1)	*type*	$\langle e_4, e_1 \rangle$
e_2	(n_{13}, n_1)	*type*	$\langle e_6, e_3 \rangle$
e_3	(n_6, n_{13})	*cite*	e_{10}
e_4	(n_7, n_3)	*type*	e_5
e_5	(n_6, n_7)	*journal*	e_8
e_6	(n_6, n_{10})	*title*	e_{12}
e_7	(n_7, n_8)	*title*	e_9

For ease of presentation, the approximate embedding shown for Query 1 only contains label approximations: `Paper` *is mapped to* `InProceedings` *and* `date` *to* `year`*. Dually, the embedding shown for Query 2 only deals with structural approximations: the data path* $\langle e_4, e_1 \rangle$ *approximates the query edge* e_1 *whereas* $\langle e_6, e_3 \rangle$ *is for the query edge* e_2*.*

It is worth noting that unbound nodes and edges are "anchored" to data elements that make the embedding possible. For instance, the embedding on the right side of Fig. 3 makes explicit the relationship between $f(3)$ *and* $f(4)$*:* `Halevy04b` *cites* `Srivastava92c`*.*

Finally, since our primary focus is on approximate graph matching, the above definition does not delve into the scoring function $s(\cdot)$ used to evaluate the IR-style operators $\ni$ and $\sim$. Our approach is general and completely application-independent and those measures that best fit the specific data and application needs can be easily integrated (e.g., IR-style TF/IDF scores, edit distance [4]).

3.3. The Ranking Model

The GEX ranking model measures answer goodness by applying a scoring function $\mathcal{S}$ to each embedding $\mathcal{E}$.[4] In line with the motivations reported in [15], we provide here a general definition for our scoring function and show the properties it satisfies. The specific scoring function used in our experiments is instead presented in Sec. 5.

$\mathcal{S}$ combines the approximations occurring at both data nodes and data edges, including query conditions, and it returns a score in $[0, 1]$ such that the higher $\mathcal{S}(\mathcal{E})$ the higher the overall approximation required for the matching.

Definition 8 (Scoring function). *Given a query* $q = (N^q, E^q, L_N^q, L_E^q, V, C)$*, an approximate embedding* $\mathcal{E}$ *for* q*, and an approximate cost function* c*, we define the score of* $\mathcal{E}$ *as follows:*

$$\mathcal{S}(\mathcal{E}) = \frac{\alpha}{\mid N^q \mid} \sum_{n \in N^q} d_L(\lambda(n), \lambda(f(n))) + \tag{1}$$

$$\frac{\beta}{2 \mid E^q \mid} \sum_{e \in E^q} \left(d_L(\lambda(e), \lambda(g(e))) + \frac{c(g(e))}{MC} \right) + \tag{2}$$

$$\frac{\gamma}{\mid C \mid} \sum_{c \in C} (1 - s(c)) \tag{3}$$

where $\alpha+\beta+\gamma=1$ *and* MC *is a normalizing constant corresponding to the maximum cost in* $\overline{SR}$*.*

Each component of $\mathcal{S}(\mathcal{E})$ contributes to the final score as follows:

[4] In the following we assume that $\overline{SR}$ has been fixed and use $\mathcal{E}$ in place of $\mathcal{E}^{\overline{SR}}$.

- the first component quantifies the semantic approximation between each query node and its corresponding data node by means of the distance function d_L applied to node labels;
- the second component quantifies both the semantic and the structural approximations between each query edge and its mapped instance in $\overline{SR}$ by exploiting d_L applied to edge labels and the approximation costs in $\overline{SR}$ to evaluate adjacency mismatches;
- the third component evaluates the query conditions by applying the inverse of the function $s(\cdot)$ used for condition evaluation.

It is worth noting that the scoring function $\mathcal{S}(\mathcal{E})$ satisfies two interesting properties:

Cost-based graph-distance semantics. Recall that $\mathcal{S}$ is applied on embeddings which refer to $\overline{SR}$. This means that the computation of the second factor of the edge approximation component of Eq. 1 can be reduced to the shortest-path problem, since the approximation cost function is monotonic and the approximation cost $c(g(e))$ for a given edge e in $\overline{SR}$ is the lowest one among all possible costs associated to alternative approximations for e.

Match-distributive semantics. The overall score of an embedding $\mathcal{E}$ can be computed in a distributive way. This means that all matching paths contribute independently to the score computation, even if they may share some common edges. This is different from other approaches which consider the single contribution of each edge only once (e.g., [5]). As already evidenced in [15], this property has important implications on the complexity of the score computation. As a matter of fact, in our model this semantics allows for the precomputation of the approximation costs in $\overline{SR}$, and these can be combined as independent parts, thus disregarding the repeated contribution of possible overlapping paths.

In GEX, given a reduced semantic relatedness relation $\overline{SR}$ over a data graph $\mathcal{G}$ and a query q, a top-k query returns the k approximate embeddings $\mathcal{E}$ with the lowest scores $\mathcal{S}(\mathcal{E})$. GEX takes advantage of the scoring function properties to efficiently support query answering and ranking, as shown in Section 5.

4. GEX Data Structures

The main objective of the GEX framework implementation is to provide efficient solutions for:

- minimizing the space required for storing graph data;
- identifying path data that is relevant according to query constraints;
- supporting a top-k retrieval algorithm that builds the best answers as soon as possible.

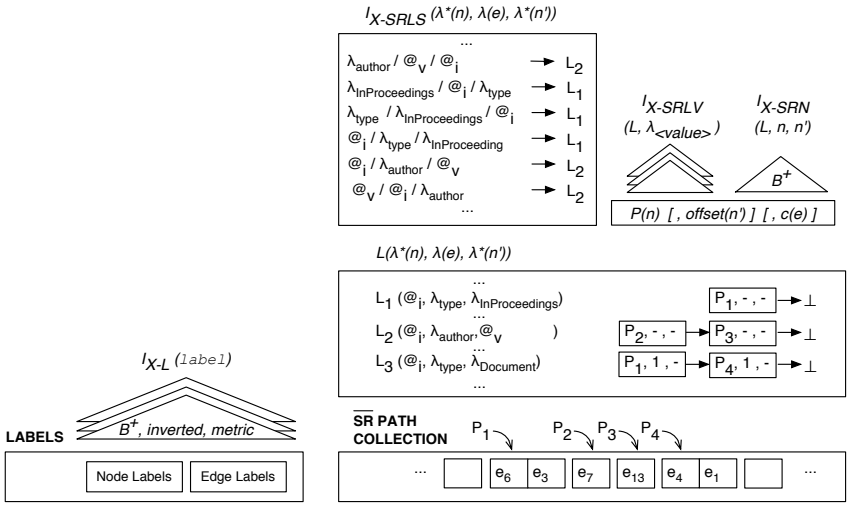


Figure 4: An overview of the GeX data structures

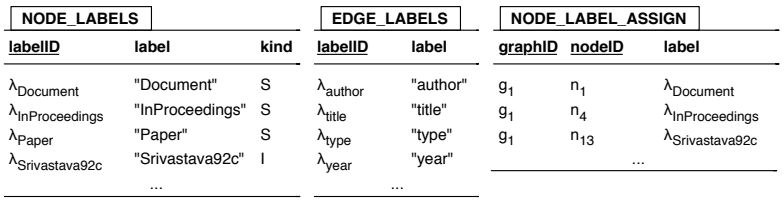
In order to achieve these goals, we implemented auxiliary structures which offer appropriate access to $\overline{SR}$ and to the graph labels. In particular, our choice has been to keep labels in a relational table and design ad-hoc data structures for $\overline{SR}$ (see Fig. 4).

4.1. Label Structures

GeX stores graph labels in relational tables according to the following schema:

```
NODE_LABELS (labelID, label, kind, dLabel)
   AK: (label)
EDGE_LABELS (labelID, label, dLabel)
   AK: (label)
NODE_LABEL_ASSIGN (graphID, nodeID, label)
   FK: label ref NODE_LABELS (labelID)
```

In particular, for each node and edge label, tables `NODE_LABELS` and `EDGE_LABELS` store, beside the actual `label` string (e.g. `Document`), a `dLabel` string containing the disambiguated label information, including the chosen sense according to the WordNet thesaurus, which is needed to determine the degree of mismatch between concept labels for approximate embedding computation (Section 3). Further, for each node label, the `kind` column indicates whether the label is a value (`V`), an instance identifier (`I`), or, in any other case, a structural information (`S`), i.e. a label which describes the structure of the data (for instance, an XML element name, the column name of a relational table, the name of an entity in RDF). Such information is straightforwardly extracted in the data graph parsing phase. In our running example, `1992` and `D. Srivastava` are value labels, `Srivastava92c` and `Halevy04b` are instance identifier labels, `Document` and `InProceedings` are structural information labels.



NODE_LABELS

labelID	label	kind
$\lambda_{\text{Document}}$	"Document"	S
$\lambda_{\text{InProceedings}}$	"InProceedings"	S
λ_{Paper}	"Paper"	S
$\lambda_{\text{Srivastava92c}}$	"Srivastava92c"	I
...		

EDGE_LABELS

labelID	label
λ_{author}	"author"
λ_{title}	"title"
λ_{type}	"type"
λ_{year}	"year"
...	

NODE_LABEL_ASSIGN

graphID	nodeID	label
g_1	n_1	$\lambda_{\text{Document}}$
g_1	n_4	$\lambda_{\text{InProceedings}}$
g_1	n_{13}	$\lambda_{\text{Srivastava92c}}$
...		

Figure 5: Label Tables

The `NODE_LABEL_ASSIGN` table assigns each node n of each stored graph $\mathcal{G}$ to its label. Since the set of edges of each graph belongs to $\overline{SR}$, edge label assignment is not stored at this level. Fig. 5 shows a portion of the content of the tables for our reference example (see also Fig. 1). Note that in GEX labels are referenced through unique numeric identifiers; however, for clarity of presentation, we represent label ids with λ and the label value as subscript and we omit the `dLabel` information.

As to the indices supporting the evaluation of conditions on value nodes, range ($=$,$>$,$<$,$\geq$,$\leq$) and containment ($\ni$) searches are supported by B+tree and inverted indices [4]. Other indices are also available for approximate searches. For instance, a metric index [16] on the disambiguated label data `dLabel` allows for label distance computations. All the different types of label indices are uniformly accessed through the requested label value `label` and are denoted in Fig. 4 as I_{X-L}; they return the identifiers λ of the labels that satisfy the specified conditions.

4.2. $\overline{SR}$ Indexing Scheme

One fundamental operation of GEX query processing is to check whether two data nodes are semantically related or not under the label constraints specified in the query. To this end, GEX employs the $\overline{SR}$ relation. Since in a connected data graph all nodes are pairwise connected by at least one path, the size of $\overline{SR}$ can potentially be very large.

An initial solution proposed in [25] was to store in a relational table the tuple (n,n',$\lambda(e)$,$p(e)$,$c(e)$) for each edge $e = (n, n') \in \overline{SR}$, with $p(e)$ the path and $c(e)$ the cost; then, a B+-tree built on the label fields was used for fast access. The large dimension of the stored table and indices and the substantial inefficiencies caused in managing query variables made this solution suboptimal, especially when matching edges containing unbound query nodes.
The redesigned indexing scheme we introduce in this paper:

- extends ad-hoc indexing solutions which have been proposed in the past for specific scenarios: keyword-based searches [15] and XML data [2, 3, 13];
- improves variable binding performances by building a concise structural summary of $\overline{SR}$;
- reduces the space requirements by separating the node pairs in $\overline{SR}$ from the related paths and by maintaining the longest paths only.

This is achieved by:

1. clustering $\overline{SR}$ on the basis of structural information, for facilitating query processing operations such as variable binding;
2. maintaining $\overline{SR}$ clusters in a two-level hierarchical organization minimizing space requirements;
3. building ad-hoc indices for the fast retrieval of the relevant portions of the clusters under specific label or node identifier constraints.

Clustering $\overline{SR}$: structural summary. Following an intuition similar to the ones behind many structural summaries (such as XML dataguides [13]), where multiple instances of repetitive portions of the graph are summarized in single index entries, the elements $(n,n',\lambda(e),p(e),c(e))$ in $\overline{SR}$ are clustered on the basis of their label triples $(\lambda(n), \lambda(e), \lambda(n'))$. In particular, the main aim is to build clusters which differentiate each other on the basis of the structural information labels only, i.e. the ones marked as `S` in the `NODE_LABELS` table. To this end, labels marked as `V` and `I` are virtually substituted with generic special labels $@_v$ and $@_i$, respectively. Each resulting cluster is then identified by a label triple $(\lambda^*(n), \lambda(e), \lambda^*(n'))$, where $\lambda^*(n)$ and $\lambda^*(n')$ can be either a label in L_C or one of the special values $@_v$ and $@_i$. For instance, the cluster $(@_i, \lambda_{type}, \lambda_{InProceedings})$ in Fig. 4 groups all the $\overline{SR}$ edges that are labeled `type` and connect any instance node to any `InProceedings` node.

Storing $\overline{SR}$: path collection and pointer lists. Due to the path-decomposition property of $\overline{SR}$, many of the paths are indeed contained one in the other. We exploit the containment relationship in order to only maintain the longest paths in $\overline{SR}$. Such paths are stored in the $\overline{SR}$ *path collection* file shown in the bottom part of Fig. 4 as a sequence of edges. On top of the $\overline{SR}$ path collection, for each cluster $(\lambda^*(n), \lambda(e), \lambda^*(n'))$, we build the $\overline{SR}$ *path pointer list* $L(\lambda^*(n), \lambda(e), \lambda^*(n'))$ (central part of Fig. 4) where each entry represents the virtual edge $e = (n, n')$ by: a pointer $P(p(e))$ to the starting edge of its path $p(e)$ in the $\overline{SR}$ path collection, the offset $offset(p(e))$ which is the number of edges to be traversed starting from $P(p(e))$ to reach the ending edge of $p(e)$, and the cost $c(e)$. The entries for each path pointer list are sorted for increasing cost $c(e)$. In each list entry, only the start pointer is mandatory. Indeed null offsets are omitted. Moreover, in the very frequent case when each edge cost $c(e)$ corresponds to the length of the path $p(e)$, we omit $c(e)$ as it coincides with $offset(p(e))$. For instance, looking at Fig. 4 and Fig. 1 and assuming that virtual edge costs are computed as the path length, the virtual edge $e = (n_{13}, n_4) \in \overline{SR}$ where $\lambda(e)$ is `type`, $p(e) = \langle e_6 \rangle$, and $c(e) = 0$ is represented in L_1 by P_1 only, which points to unary path e_6 in the $\overline{SR}$ path collection. A longer path containing the above one is pointed by the first element of L_3 which represents the virtual edge $e = (n_{13}, n_1) \in \overline{SR}$ where $\lambda(e)$ is `type`, $p(e) = \langle e_6, e_3 \rangle$, and $c(e) = 1$.

Rotated lexicon and content indices for fast retrieval. GeX data structures also include specific indices, namely I_{X-SRLS}, I_{X-SRLV}, and I_{X-SRN} (Fig. 4), for fast retrieval of the relevant graph data under constraints on nodes and edges. More precisely, I_{X-SRLS} is an inverted index on the rotated

lexicon of the $\overline{SR}$ structural summary, allowing a very fast identification of the path pointer lists on the basis of query labels. The structure is derived from the XML "rotated lexicon" presented in [2]. In our case, the lexicon is given by the set of label tuples $(\lambda^*(n), \lambda(e), \lambda^*(n'))$ and each entry of the rotated lexicon is expressed as a sequence of the three labels of the tuples, i.e. $\lambda^*(n)/\lambda(e)/\lambda^*(n')$. Then, the lexicon is "rotated", i.e. all the possible rotations of the entries are generated. The three rotated entries originating from the same tuple $(\lambda^*(n), \lambda(e), \lambda^*(n'))$ point to the same $\overline{SR}$ path pointer list $L(\lambda^*(n), \lambda(e), \lambda^*(n'))$. For instance, for the sequence $@_i/\lambda_{type}/\lambda_{InProceedings}$ in Fig. 4:

$$\left.\begin{array}{l} @_i/\lambda_{type}/\lambda_{InProceedings} \\ \lambda_{InProceedings}/@_i/\lambda_{type} \\ \lambda_{type}/\lambda_{InProceedings}/@_i \end{array}\right\} \longrightarrow L_1$$

Thanks to the properties of rotated lexicons [2], search operations will consists of simple binary searches on I_{X-SRLS}, and all the relevant matching lists will be found close together after the first match (see Section 5). Note that such a rotation preserves the edge direction, but not the role of the involved labels (i.e. node vs edge labels). To preserve this information, we explicitly distinguish edge labels from node labels by adding a special character in front of the label (for simplicity of presentation, we omit this notation in our discussion). As we will see in the following section, I_{X-SRLS} is particularly useful when dealing with variables.

Finally, the I_{X-SRLV} and I_{X-SRN} indices provide fast retrieval of $\overline{SR}$ elements in a given list L in response to specific search values and node identifiers, respectively. To be more precise, the indexing fields of I_{X-SRLV} are a list L and a value label identifier $\lambda_{<value>}$. Instead, I_{X-SRN} is accessed through a list L and the pair of node identifiers (n, n') to be searched in $\overline{SR}$. I_{X-SRN} is implemented as a set of B+trees, while different kinds of predicates are supported in I_{X-SRLV} by means of B+tree (for range predicates), inverted (containment predicates) or metric (similarity predicates) indices.

5. Top-k Query Answering Algorithm

In this section we introduce the GeX top-k query answering algorithm, that efficiently generates the top-k answers according to the scoring function $\mathcal{S}(\mathcal{E})$.

Similarly to other proposals for top-k graph query processing [15, 30], our algorithm founds on the principles of the Threshold Algorithm (TA) [9] which has been proven optimal in terms of number of visited items. Specifically, the algorithm follows two main phases:

1. *Cursor initialization*, where the query edges are assigned to cursors which fetch data from the relevant path pointer lists;
2. *Cursor access and solution building*, where sorted access in parallel is performed on the cursors and the most relevant solutions are built.

However, the GEX querying algorithm operates in a more challenging scenario than the other TA-derived algorithms because of its approximate matching approach that includes unbounded nodes and edges, label approximation and undirected edges. Such features affect both phases of the algorithm and we introduced significant modifications to the TA in order to efficiently deal with them.

In the cursor initialization phase, cursors can be associated to more edge lists. In particular, the number of such lists can be very high in case of unspecified labels. The algorithm limits the number of relevant lists by accessing the newly introduced index I_{X-SRLS} where data is organized according to structural information only. This represents a definite advantage over our past proposal [25], where we had to open many independent cursors for each possible values and/or instances matching an unspecified label. Moreover, by means of the rotated lexicon, such a phase is able to quickly identify queries having no structural matches.

In the cursor access and solution binding phase, each object in one cursor conceptually joins with more than one object in each of the others. Therefore, all answers involving such object should be computed and they can be potentially a large number. For this reason, a pruning threshold is applied not only to decide if the generated answers are sufficient, but also during answer computation to avoid completing the computation of useless answers. Further, as far as the cursor access is involved, differently from [15] where only round-robin accesses (`ROUND_ROBIN`) are considered, GEX includes additional cursor selection strategies whose goal is to try to build better ranked answers as soon as possible.

All such benefits will be clearly demonstrated by the experimental results and comparisons shown in Section 6.

The algorithm is detailed in the following by considering the case of a single graph; otherwise, it can be easily extended to check graph identifiers. Moreover, for the sake of clarity, we start by considering the case of exact matching for node and edge labels, therefore, $\mathcal{S}(\mathcal{E}) = \frac{\beta}{2|E|} \sum_{e \in E} \frac{c(g(e))}{MC}$; then, we show how the algorithm can be easily extended to deal with approximate labels (Section 5.3).

5.1. Cursor initialization

Let $q = (N, E, L_N, L_E, V, C)$ be a query and let firstly suppose that all the query labels are specified and all the query edges are directed. In this simple case, each query edge $e_i = (n_{S(i)}, n_{E(i)})$, for $i \in [1, |E|]$, is associated with a cursor C_i that iterates on the elements in the sorted list $L^i \equiv L(\lambda^*(n_{S(i)}), \lambda(e_i), \lambda^*(n_{E(i)}))$; $S(i)$ and $E(i)$ denote the ids of the start and end nodes of e_i, respectively. The cursor is easily initialized by: (a) searching for the involved label ids in I_{X-L} according to the nodes and edge labels $\lambda(n_{S(i)})$, $\lambda(e_i)$, $\lambda(n_{E(i)})$; (b) accessing I_{X-SRLS} in order to retrieve list L^i.

When an edge e_i is undirected or contains variables, its cursor C_i will possibly be associated with more than one list, i.e. $L^{i:1}$, $L^{i:2}$, ..., $L^{i:t}$. More

specifically, for an undirected edge, the accessed lists would simply be $L^{i:1} \equiv L(\lambda^*(n_{S(i)}), \lambda(e_i), \lambda^*(n_{E(i)}))$ and $L^{i:2} \equiv L(\lambda^*(n_{E(i)}), \lambda(e_i), \lambda^*(n_{S(i)}))$. In this case, the cursor still offers a sorted access to the elements in $L^{i:j}$, j=1 . . . t, by accessing the t sorted lists in parallel and advancing on the list having the lowest cost element.

On the other hand, the case of unbound nodes and edges is efficiently dealt with by means of the I_{X-SRLS} rotated lexicon features. More specifically, query edges containing variables are processed by substituting each variable *var* with the wildcard @, and by rotating their sequences so that special symbols appear as last. For instance, an edge $\$x/\lambda_{type}/\lambda_{Journal}$ is transformed to $\lambda_{type}/\lambda_{Journal}/@$. In the index, the wildcard @ will match with any label, $@_i$, and $@_v$. Then, the transformed sequence is used to search I_{X-SRLS} through a simple binary search for finding the first match. Because of how I_{X-SRLS} is constructed, possible additional matches are then available in the immediately subsequent entries. In case the variable *var* appears in a query predicate $var\langle op\rangle v$, we access I_{X-SRLS} by substituting *var* with $@_v$, and we restrict the selected list $L^{i:j}$ by accessing I_{X-SRLV} through ($L^{i:j}$,λ_v).

With reference to the complete top-k query answering algorithm shown in Fig. 6, cursor initialization is specified at lines 1-4.

Example 5.1. *Going back to our data graph (Fig. 1) and Query 1 (Fig. 2), we simplify the query to prevent the case of approximate label matching, a feature which is not essential to understand the core of the algorithm. Let us consider only the first three query nodes (n_1, n_2 and n_3) and modify the labels of n_1 and n_3 to* `InProceedings`*, and* `D. Srivastava`*, respectively. Cursors are initialized in the following way:*

- *the label ids matching* `InProceedings`*,* `D. Srivastava`*,* `type`*, and* `author` *are found by means of I_{X-L};*
- *$\overline{SR}$ is then queried through I_{X-SRLS} and I_{X-SRLV}: the first edge e_1 contains a variable $\$w$, so, first of all, the sequence $\$w/\lambda_{type}/\lambda_{InProceedings}$ is rotated to $\lambda_{type}/\ \lambda_{InProceedings}/\ \$w$, then I_{X-SRLS} is searched through the rotated sequence. From Fig. 4, we see that the only matching list is L_1, and, since e_1 does not involves values, cursor C_1 is initialized to the whole L_1;*
- *in a similar way, L_2 is associated to edge e_2 by means of I_{X-SRLS}. However, since e_2 involves a value, index I_{X-SRLV} is accessed through (L_2, $\lambda_{D.Srivastava}$) and cursor C_2 is initialized to the elements of L_2 containing* `D. Srivastava` *as a value.*

Each cursor C_i is accessible through the following functions:

- next() advances the cursor and returns the next cursor element $(n, n', c(e))$, i.e. the next node pair $e = (n, n')$ together with its cost $c(e)$;
- seek(n,n') performs indexed random accesses on the cursor through the node identifiers (n, n') and returns the matching cursor elements (only one or both ids can be specified);

Algorithm 1 answerQuery(q,k)

1: **for all** $i \in [1, |E|]$ **do** *// Phase 1: Cursor initialization on query edges*
2: $((\lambda(n_{S(i)}), \lambda(e_i), \lambda(n_{E(i)})) \leftarrow I_{X-L}$(labels in e_i)
3: $\{L^{i:j}\} \leftarrow I_{X-SRLS,SRLV}(\lambda^*(n_{S(i)}), \lambda(e_i), \lambda^*(n_{E(i)}), \lambda_{<value>})$
4: $C_i \leftarrow$ newCursor($\{L^{i:j}\}$)
5: i=0; $Ans = \emptyset$;
6: **while** $(i \leftarrow$ getNextCursor(i)$) > 0$ **do** *// Phase 2: Cursor access and solution building*
7: $(n, n', c(e)) \leftarrow C_i$.next()
8: computeAnswers($i, n, n', c(e), 0, \emptyset, \emptyset, Ans$)
9: $lBound \leftarrow \sum_{j=1}^{|E|} C_j$.peekCost()
10: **if** $|Ans| \geq k$ and $lBound \geq Ans[k].score$ **then**
11: abort answer computation
12: output Ans

Figure 6: Top-k query answering algorithm

- curCost() returns the cost c(e) of the current element, i.e. the element on which the cursor is currently positioned;
- peekCost() returns the cost $c(e)$ of the next cursor element (the cursor position is not advanced);
- size() returns the number of elements following the current position.

5.2. Cursor access and solution building

In line with the TA principles [9], Algorithm 1 computes the k best answers by performing a parallel sorted access to the $|E|$ cursors (lines 5-12). Specifically, a starting cursor C_i is selected at line 6 through the getNextCursor algorithm detailed in Fig. 8, C_i advances through the next() function (line 7), then, in line 8 algorithm `computeAnswers` performs random access to the other cursors and computes (some of) the answers originating from e. The top-k answers computed so far are maintained in list Ans sorted on increasing $score$ values.

After each answer computation step, the algorithm computes the score $lBound$ of the set of the next node pairs under sorted access to the $|E|$ cursors as they were a solution, and stops the process whenever at least k answers have been seen whose grade is smaller than $lBound$ (lines 9-11). Indeed, as cursor elements are ordered by increasing cost, $lBound$ represents the best score of any solution which could be computed in the following steps.

Example 5.2. *Continuing our example, after cursor initialization let us suppose that cursor selection prioritizes access to the most selective cursors:*

- *sorted access is performed on C_2 and the node pair (n_{13},n_{11}) is extracted from the cursor;*
- *all answers involving the extracted pair are efficiently constructed by performing random accesses in the other cursors; for the first query edge, index I_{X-SRN} is queried for $n = n_{13}$ (being this the starting node for edge e_1);*

Algorithm 2 computeAnswers($i, n, n', cost, curS$, $nList$, $eList$, Ans)

```
1: nList[S(i)] = n
2: nList[E(i)] = n'
3: update(eList, i, n, n')
4: curS ← curS + cost
5: ī ← getNextQueryEdge(i)
6: if ī < 0 then      // answer completed
7:    if Ans[k].score > curS OR |Ans| < k then
8:       Ans.add((nList, eList, curS))
9:    return
10: if |Ans| ≥ k ∧ curS + C_ī.peekCost()) ≥ Ans[k].score then      // abort computation of current answer
11:    return
12: else      // continue answer computation recursively
13:    while ((n̄, n̄', cost̄) ← C_ī.seek(eList[ī].n_S, eList[ī].n_E)) ≠ NULL do
14:       computeAnswers(ī, n̄, n̄', cost̄, curS, nList, eList, Ans)
15:    return
```

Figure 7: Answer computation algorithm

- *then, results are generated by combining the matches for each of the query edges; in our simple case, exactly one answer is found, i.e.* $\mathcal{E}^{\overline{SR}} = (f, g)$*, where* $4 = f(1)$*,* $13 = f(2)$ *and* $11 = f(3)$ *and with the edge assignment defined as:*

e	$g(e)$	$\lambda(g(e))$	$p(g(e))$
e_1	(n_{13}, n_4)	*type*	e_6
e_2	(n_{13}, n_{11})	*author*	e_{13}

  Finally, the score of the answer is computed (in this case, it is 0 *since no approximations are performed), and, since the cursors do not contain additional data, the algorithm stops and returns the results.*

As far as `computeAnswers` is concerned (see Fig. 7), an answer is a triple $(nList, eList, score)$ where $nList[1, \ldots, |N|]$ is a list of data nodes, one for each query node, which encodes the node-assignment function f, $eList[1, \ldots, |E|]$ is a list of node pairs (n_S, n_E) in $\overline{SR}$, one for each query edge, which encodes the edge-assignment function g, and $score$ is the score $S(\mathcal{E})$ of the approximate embedding $\mathcal{E} = (f, g)$. In the following, we will denote as $eList[i].n_S$ and $eList[i].n_E$ the start and end nodes of the i-th node pair stored in $eList$, respectively.

At the beginning, the entries of both lists are initialized with node placeholders but those entries corresponding to (n, n') (lines 1-3) and then they are updated while recursively computing the answer. More precisely, each time a data node pair (n, n') is added to the current solution, the update() function updates $eList[i]$ and uses n and n' to update the entries corresponding to e_i's adjacent edges.

Algorithm 3 getNextCursor(*current*)

```
1: next ← −1
2: if Mode =ROUND_ROBIN then
3:     next ← pick i from [1, ..., |E|] in round robin starting from current such that C_i.size()> 0
4: else if Mode =NEXT_BEST then
5:     next ← find i ∈ [1, ..., |E|] minimizing (C_i.peekCost() −C_i.curCost()) such that C_i.size()> 0
6: else if Mode =MAX_SEL then
7:     next ← find i ∈ [1, ..., |E|] minimizing C_i.size() such that C_i.size()> 0
8: return next
```

Figure 8: Cursor selection algorithm

Each `computeAnswers` call (line 14) is led by the cursor $C_{\overline{i}}$ associated with the next unvisited query edge which has been (partially) bound as an update() side effect (line 5).

Whenever the answer computation process ends, the answer is added to Ans if Ans contains less than k entries or if its score is no greater than the k-th one (lines 6-9).[5] Otherwise, the process can be interrupted for two reasons: either the lower bound of the answers which should be computed exceeds the pruning threshold $Ans[k].score$ (lines 10-11) or no matching $\overline{SR}$ edge in the selected cursor $C_{\overline{i}}$ is found (lines 13-15). In particular, $C_{\overline{i}}$.seek() performs indexed random accesses on its elements through the node identifiers available in $eList[\overline{i}]$.

As far as cursor selection is concerned, algorithm `getNextCursor` shown in Fig. 8 returns the next cursor according to different selection strategies. Besides round-robin accesses (`ROUND_ROBIN`) which is typically available in the literature [15], the algorithm includes the following additional cursor selection strategies. The `NEXT_BEST` picks an edge from the cursor that minimizes the difference between the cost of the next (i.e. upcoming) element and that of the current one. In this case, the idea is to minimize the lower bound of the score of the potential solutions that will be generated next, i.e. the ones that will originate from the picked edge. Moreover, as list sizes often show a regardable variance, we can start building answers from the most selective edges by prioritizing the cursors whose size is the smallest one (`MAX_SEL`). For instance, consider those queries where some of the nodes are left unbound: edges containing such nodes are not a good choice to start processing with and, because of their very low selectivity, they will be among the last ones to be selected.

5.3. Dealing with Approximate Labels

When dealing with approximate label, each query label is possibly associated with more than one data label, i.e. those labels similar to the query label w.r.t. d_L.

The GEX querying algorithm presented so far is straightforwardly extended to deal with this case thanks to the way cursors are initialized and accessed. Indeed, any query edge cursor can be associated to more than one list, where the objects of each list share the same label.

[5] Note that for the sake of simplicity score normalization is omitted.

Query	#n	#e	#any	#exp ans	Description	Struct relax	(w/ $ edge)	Label approx
					Russia dataset			
R1	3	2	1	1	"The authors that studied at the University of Kazan"			
R2	3	2	1	4	"*People that studied* at the University of Kazan"	x		
R3	3	2	1	4	"*People connected* ($ edge) to University of Kazan"	x	x	
R4	4	3	2	4	"The *public spaces connected* ($ edge) to Lev Tolstoj"	x	x	
R5	7	6	3	4	"The *public spaces that lie* in the same town as Vosstaniye Square"	x		
R6	3	2	1	4	"*Human beings that frequented* the University of Kazan"	x		x
R7	7	6	3	4	"*Places that are in* the same city as Vosstaniye Square"	x		x
Query	**#n**	**#e**	**#any**	**#exp ans**	**Description**	**Struct relax**	**(w/ $ edge)**	**Label approx**
					DBLP-S and DBLP datasets (expected answers are given for DBLP-S)			
Q1	3	2	1	12	"Articles of year 1997"			
Q2	3	2	1	36	"*Papers (generically) of year* 1997"	x		x
Q3	5	4	3	10	"Titles of *papers of year* 1990 and *connected* ($ edge) to STACS conference"	x	x	x
Q4	6	6	4	148	"Titles of *documents created by a person* who has also *created a document* in 1994"	x		
Q5	5	4	3	10	"Name of *works of* 1990 and *connected* ($ edge) to STACS conference"	x	x	x

Table 1: Query specifications for Russia, DBLP-S and DBLP datasets

Thus, fixed a query edge $e_i = (n_{S(i)}, n_{E(i)})$, it is sufficient for its cursor C_i to associate each list $L^{i:j}$ with three values, $D_S^{L^{i:j}}$, $D_E^{L^{i:j}}$ and $D_e^{L^{i:j}}$, expressing the distance of the starting node, ending node and edge labels from the query ones, respectively. Such information is combined with the cost information of the list elements so that the cursor still offers sorted data access. As to answer ranking, $nList$ and $eList$ entries are simply extended with appropriate d_L values in order to compute the overall score of each answer. Note that the top-k algorithm is still correct being sorted data access guaranteed also in this general case.

6. Experimental Evaluation

We contextualized GeX through the RDF-like instantiation of SR shown in Appendix, and we performed a thorough analysis of both the effectiveness and the efficiency of GeX on several RDF-based real world datasets. In this section we present a selection of the most significant results we obtained, including specific tests comparing our effectiveness figures with the ones obtainable by different state of the art approaches and detailed time and space efficiency comparisons between different alternative implementations of our system.

6.1. Experimental Setting

Among the several collections we employed to test GeX, we selected Russia[6], DBLP-S and DBLP for the following discussion, as they allow us to completely stress the system from all the required perspectives. Russia describes several information about the country's cities and people, while DBLP-S (small version) and DBLP (complete version) are extracted from the well known DBLP scientific bibliography data. Note that DBLP data is also one of the most exploited resources in the literature for graph search testing purposes [15, 30, 23]. The structure of such graphs is quite complex: in particular, the Russia dataset

[6]Publicly available at http://www.rdfdata.org

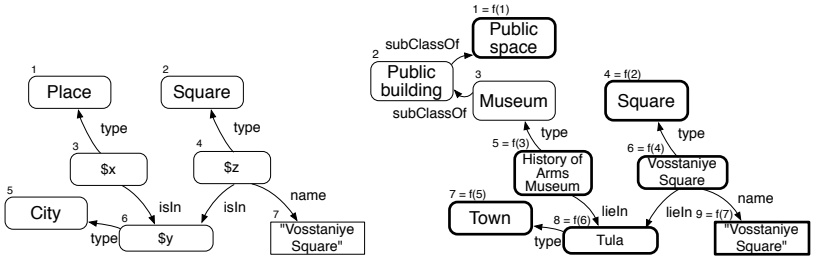


Figure 9: One of the Russia queries (R7, left), together with one possible embedding (right)

has a very detailed schema, with over 500 different and richly interconnected concepts and properties, while DBLP and DBLP-S build from a typical data-centric scenario of nearly 100 schema elements and create a dense bibliography network of relations between authors and their works. Russia and DBLP-S, which are not very big in size (1012 nodes / 1613 edges and 4373 nodes / 7779 edges, respectively), will be employed for testing the system effectiveness. On the other hand, the very large DBLP collection will stress the approach also from an efficiency point of view, with 1897225 nodes and 5009848 edges.

For all collections, we created a set of significant queries, named R1 to R7 (Russia) and Q1 to Q5 (DBLP and DBLP-S). Table 1 shows an overview of the employed queries (number of nodes, number of edges, number of "any-label" i.e. variables \$, number of expected answers), together with their textual descriptions and the specific challenges they present to be correctly solved, including:

- structural approximations (with or without generic semantic connection (\$) edges), where the structure of the original schema needs to be relaxed. The relevant part of the query is highlighted in italics in the Table 1 description;
- semantic label approximations, where some of the nodes do not employ the same vocabulary of the schemas. Relevant terms are underlined in Table 1;
- a combination of both.

Besides the simple R1, queries R2 to R5 are gradually more complex and require different approximations: queries R2 and R3 generically ask for "people" (a concept subsuming a large number of more specific classes), similarly to queries R4 and R5 asking for generic "public spaces", while queries R3 and R4 employ generic semantic connection (\$) edges. On the other hand, queries R6 and R7 are designed to stress the semantic label similarity engine and employ very generic and alternative terms (different from the schema vocabulary and the other queries) on purpose. Figure 9 shows, as illustrative example, the complete graph representation (and, on the right, a possible embedding) of R7, while the others can be easily inferred from the descriptions. For instance, label approximations are needed for both nodes ("Place", "City") and edges

	GeX				Exact			Web	Naive		NAGA MP		TALE MP	
Query	P	recall	P(lowT)	P@ea	#q	P	recall	#q	P	recall	P	recall	P	recall
Russia dataset														
R1	1	1	1	1	1	1	1	n/a	0,200	1	1	1	1	1
R2	1	1	0,667	1	13	n/a	n/a	n/a	0,308	1	1	1	1	1
R3	1	1	0,571	1	689	n/a	n/a	n/a	0,308	1	0,31	1	0,4	1
R4	1	1	0,8	1	637	n/a	n/a	n/a	0,044	1	0,04	1	0,21	0,5
R5	1	1	0,5	1	49	n/a	n/a	n/a	0,267	1	1	1	1	0,75
R6	0,8	1	0,363	1	13	n/a	n/a	n/a	0,308	1	n/a	n/a	0,8	1
R7	0,8	1	0,44	1	637	n/a	n/a	n/a	0,044	1	n/a	n/a	0,21	0,5
DBLP-S Dataset														
Q1	1	1	0,357	1	1	1	1	3	0,002	1	1	1	1	1
Q2	1	1	1	1	4	n/a	n/a	2	0,061	1	n/a	n/a	1	1
Q3	1	1	0,909	1	25	n/a	n/a	n/a	0	1	n/a	n/a	0,01	1
Q4	1	0,94	0,754	1	64	n/a	n/a	41	0	1	1	0,93	1	0,94
Q5	0,91	1	0,667	1	25	n/a	n/a	n/a	0	1	n/a	n/a	0,01	1

Table 2: Effectiveness results comparison - Russia and DBLP-S

("isIn"), while structural approximations are exploited to derive the semantic connections between the nodes $f(1)$ and $f(3)$. Following the same principles, the DBLP queries start from similar ones already exploited in literature (e.g. in [15, 30]) but are enhanced so to exploit the GeX peculiarities (e.g. "document" subsumes a large number of more specific classes, "paper" and "work" are not in the schema vocabulary, and so on).

In order to apply label similarities, the concept labels are disambiguated with the STRIDER [26] structural disambiguation system; the data labels having a similarity higher than a specified threshold are associated to the query labels. The system, including full data structures and algorithms, is fully implemented in Java 1.6. The reference implementation which we employed for most of our tests is an ad-hoc solution exploiting the data structures described in Section 4.2 (which we will refer to as "enhanced"), completely built from scratch in java low-level programming. Different GeX implementations are also considered and compared in specific sections: together with the "enhanced ad-hoc" solution, we consider the "enhanced RDBMS" one based on a full relational system (PostgreSQL 8.4) and "enhanced DBLib", based on a light and embeddable database library (Oracle BerkeleyDB 3.2 Java Edition). Further, as a baseline, we also compare the "DBLib" implementation and index structures as described in [25]. All the experiments are executed on an Intel Core2 Quad Q9450 2.66GHz Windows 7 64 bit workstation, equipped with 4GB RAM and a 500GB SATA II hard drive.

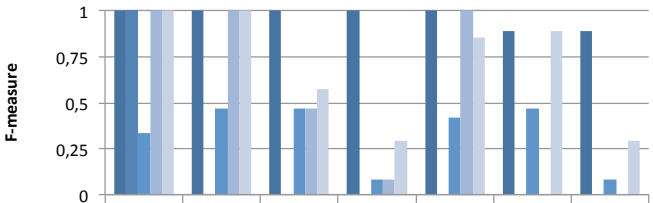
6.2. Effectiveness Evaluation and Comparison

Table 2 presents a detailed summary of the effectiveness results we obtained for Russia (upper part of the table) and DBLP-S (lower part). The table shows different measures and comparisons: the precision (i.e. percentage of relevant retrieved answers w.r.t. the retrieved ones) and recall (i.e. percentage of relevant retrieved answers w.r.t. existing relevant ones) levels achieved by GeX

in a standard and low similarity threshold settings (more on the latter at the end of this section), comparing them with the ones achievable through different approaches: we simulated an exact matching approach and a naïve structural approach (similar to [6]). While the exact match approach clearly has a very limited flexibility, the considered naïve approach is somehow opposed: the naïve results are the ones that would have been retrieved by computing all node matches and connecting them in the data in all possible ways, thus disregarding our SR information. For DBLP we also considered the specific Web portal search opportunity, not available for Russia, and, where applicable, we also analyzed the number of queries ($\#q$) that would be necessary to obtain all the expected results (as opposed to one single query for our system). Further, we simulated all and only the aspects of the NAGA [19] and TALE [29] state of the art proposals which are relevant to our effectiveness evaluation analysis, in particular their query model and matching paradigms (NAGA MP and TALE MP in table). Please note that not all the considered approaches support all the features required to correctly answer the queries (this explains the various "n/a" in the table): for instance, while NAGA is a semantic approach and is thus correctly able to handle `isA` hierarchy navigation, TALE only offers a syntactical "degree of approximation" for relaxing structures. On the other hand, NAGA has no support for semantic label approximation, while TALE has a syntactic graph matching technique which is able to cope with label mismatches by "plugging-in" external label similarities: in this case, for a fair comparison with GeX, we exploited our semantic similarity. The same applies to the NAGA scoring model, which is not directly applicable to our scenario (for a web-based corpus, such as the knowledge base they use, it would depend on the informativeness and confidence of the authoritative pages from which it is extracted).

Let us start our analysis by examining the results for Russia. Query R1 is the most simple, since it does not require approximations, thus the exact, NAGA, TALE and GeX approaches return the correct answer, Lev Tolstoj, with a precision and recall of 1. The naïve approach, however, builds a larger number of answers, since it connects other authors to the required University through paths that are not semantically relevant (precision 0.2). Note that, even if the TALE approach is not able to assess the semantic relevance of the paths, in this case it is able to prune out the wrong answers thanks to the syntactical "degree of approximation" threshold (the corresponding embedding structures would be too different from the query one). For all queries (R1-R7) we achieve very good (perfect for R1-R5) precision; this is made possible by exploiting SR and, for the label similarity computations, a "safely" high similarity threshold setting, nonetheless also allowing us to obtain perfect recall levels. Instead, the exact and naïve approaches are either not applicable (no retrieved exact matches) or very inaccurate. Also, note that "rewriting" our query to all possible exact queries is almost infeasible due to the excessive growth of the number of exact queries that would be required (for instance, more than 600 queries to be processed for R3 and R4).

Let us now focus on the NAGA and TALE performances, which are not



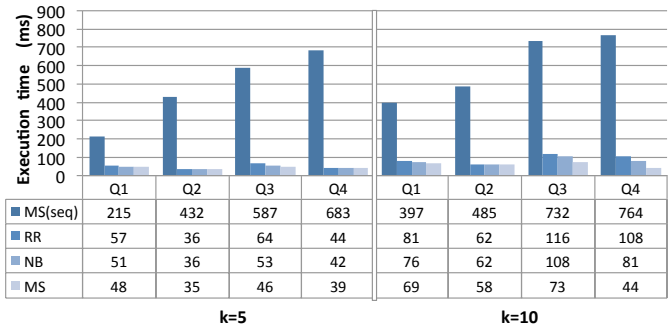
	R1	R2	R3	R4	R5	R6	R7
GeX	1	1	1	1	1	0,89	0,89
Exact	1	0	0	0	0	0	0
Naive	0,33	0,47	0,47	0,09	0,42	0,47	0,08
NAGA MP	1	1	0,47	0,08	1	0	0
TALE MP	1	1	0,57	0,30	0,86	0,89	0,30

Figure 10: F-measure results comparison - Russia

always able to deliver satisfying precision and recall levels for the considered queries. This is due to the specific features and ways of working of the two systems: NAGA correctly handles the semantics in R2 and R5, however generic semantic connection ($) edges are supported only by means of the "connect" edge, thus disregarding semantic information and producing a large number of useless answers (for instance, precision 0.308 and 0.044 for queries R3 and R4, respectively). Further, queries with semantic label approximation, such as R6 and R7, can not be processed by such system. On the other hand, in TALE the offered syntactical "degree of approximation" does not appear to be able to discriminate good and bad results for all queries requiring semantic-aware structural approximations, thus producing irrelevant answers (low precision in R3, R4 and R7) and missing relevant answers (low recall in R4, R5 and R7). Fig. 10 presents a visual summary of the effectiveness results achieved by the different considered techniques by showing the achieved F-measures (weighted harmonic means of precision and recall): our results show an F-measure of 0.89 or higher for all the Russia queries.

These good results are also confirmed by the DBLP queries (lower part of Table 2). Again, differently from the other systems, GeX achieves the highest precision and recall levels. In this case, we can also see that accessing the DBLP Web portal would require a significant work from the user who should submit a possibly very large number of queries to retrieve the expected results (see the $\#q$ column).

Finally, we present a small but representative sample of a specific effectiveness evaluation we performed on our ranking model and function. In particular, we simulated a more rich and "noisy" answer set to Russia and DBLP queries by significantly lowering the semantic approximation threshold employed for label match and, together with precision P (denoted as $P(lowT)$ in Table 2), we computed precision at recall level ea, $P@ea$, where ea is the number of expected answers of each query. As shown in table, even if $P(lowT)$ is globally lower for the whole retrieved answer set (for instance, "title" is now also similar to label "note"), the function proves to be effective in discriminating the irrelevant



	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4
MS(seq)	215	432	587	683	397	485	732	764
RR	57	36	64	44	81	62	116	108
NB	51	36	53	42	76	62	108	81
MS	48	35	46	39	69	58	73	44

Figure 11: Query Execution time for DBLP

	%compl answers			#sorted accesses			#random accesses		
Query	RR	NB	MS	RR	NB	MS	RR	NB	MS
Q1	35%	64%	71%	14	11	7	7	6	5
Q2	50%	78%	98%	10	7	5	7	6	5
Q3	29%	26%	45%	17	23	16	21	17	15
Q4	29%	45%	97%	17	13	3	145	98	72

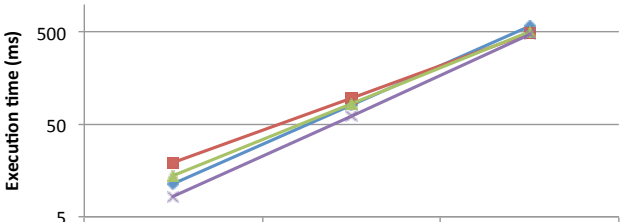
Table 3: Details on performed index accesses for DBLP

answers and keeping them low in the ranking, with perfect $P@ea$ levels.

6.3. Efficiency Evaluation and Scalability

Along with a good effectiveness, a graph query answering approach also necessarily needs to be very efficient in order to be useful, since the size of the managed graphs can be very high. In the following tests, we analyze the performance of GEX on the large DBLP collection, considering query execution time and the number of index accesses performed by the different available cursor selection strategies and access modes. In this section, we will consider the performance of queries Q1-Q4, while query Q5 delivers a performance very close the Q3 one and is therefore not shown. Further, specific query variations will be exploited to test the system scalability. All the time figures we show are those obtained by our "enhanced ad-hoc" implementations. A detailed comparison of the performances offered by our different implementations, including full indexing space requirements analysis, will be given in Section 6.4.

Fig. 11 shows the execution time of queries Q1 to Q4 for the Round Robin (RR), Next Best (NB) and Max Sel (MS) cursor selection strategies, for both $k = 5$ and $k = 10$. For all of them, the random accesses performed by the seek() function exploit indices; the sequential versions of these strategies proved significantly slower then their index-based counterparts, thus we present only the best performing one (denoted as "MS(seq)" in figure) as a baseline. First of all, we can see that the index-based execution time, even for the most complex queries, is low, less than 0.1 seconds for this large dataset ($k = 10$), except for Q3 and Q4 that, in some cases, slightly exceed this value. As to the cursor selection methods, RR generally proves to be the worse performing strategy



	k=1	k=10	k=100
Q6	11	81	579
Q7	19	98	498
Q8	14	83	502
Q9	8	62	471

Figure 12: Scalability test results for DBLP (MS strategy)

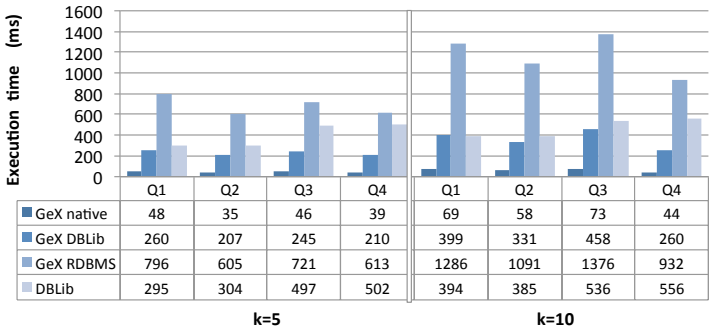
and is particularly outperformed by the others in case of queries having many edges with different selectivity (such as Q3 and Q4). MS strategy is the most efficient in all situations (less than 80 milliseconds); NB performance is close but equally satisfying, also considering that, differently from MS, it does not require to know the cursors' size to work, thus possibly proving more versatile. MS scalability w.r.t. k also proves the most encouraging for all queries, with a computation time increase well under 60% in going from $k = 5$ to $k = 10$.

Table 3 completes the picture by comparing the different strategies, for $k = 5$, in terms of the percentage of completed answer computations and the number of required disk accesses, both sorted and random. Notice the high percentage of completed answer computations, specifically for MS, meaning that the time spent in starting useless answer computations is minimized. Further, the MS strategy also provides the lowest number of disk accesses, thus justifying the previously examined figures.

Finally, we deepen the scalability analysis by stressing GeX with high values of k: Fig. 12 shows the figures obtained by the MS strategy for k from 1 to 100 on queries Q6-Q9, which are derived from query Q1 by considering query values with lower selectivity, so to obtain a number of answers which is significant for the analysis. As we can see, the encouraging trends discussed for Q1-Q4 for lower k values are confirmed.

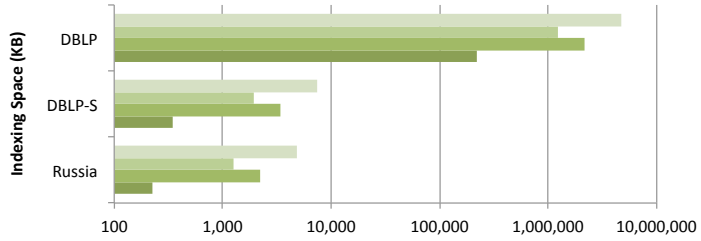
6.4. Time and Space Performance Comparison between Different Implementations

Up to now we considered the performances of GeX "enhanced ad-hoc" implementation. We now conclude the experimental evaluation with a comparison of the efficiency of the different implementations of our system, i.e. we will also consider the alternative "enhanced RDBMS" and "enhanced DBLib" versions. All such systems are based on the enhanced index structures and algorithm presented in this paper. Further, as a baseline, we will also consider the original system and index structures ("DBLib"), as presented in [25]. Let us start by examining query execution time for Q1-Q4 for DBLP, as shown in Fig. 13:



	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4
GeX native	48	35	46	39	69	58	73	44
GeX DBLib	260	207	245	210	399	331	458	260
GeX RDBMS	796	605	721	613	1286	1091	1376	932
DBLib	295	304	497	502	394	385	536	556

Figure 13: Performances of different implementations for DBLP



	Russia	DBLP-S	DBLP
DBLib	4,825	7,488	4,680,000
GeX RDBMS	1,251	1,941	1,213,200
GeX DBLib	2,201	3,416	2,134,800
GeX native	225	349	218,000

Figure 14: Indexing space comparison between different implementations

as we can see, the performances are in general satisfying (typically well under one second for each query) for all implementations, however the solutions based on external data engines, such as DBMSs and embedded Database Libraries, require a significant time overhead w.r.t. the ad-hoc solution we programmed in ad-hoc java code. Moreover, another interesting comparison is between the "enhanced DBLib" and "DBLib" versions: in this case both solutions are based on the Berkeley DB library and thus we can better appreciate the efficiency improvements given by the novel index structures and algorithms presented in this paper w.r.t. the ones in [25]. Finally, Fig. 14 completes the picture with a detailed comparison of the indexing space requirements of all the considered implementations: first of all, notice that, as we expected, the enhanced index structure optimizations allow us to drastically reduce the space requirements of the original "DBLib" version. This is true for all the enhanced implementations: for instance, for the large DBLP collection, the "enhanced DBLib" and "enhanced RDBMS" reduce space occupation to 45% and 25% of the original requirements, respectively. Instead, the "enhanced ad-hoc" reveals by far the most advantageous solution also from the space requirements point of view: for all datasets, the indexing space is reduced to as little as 4% of the original

requirements.

7. Related Work

The approximate matching of complex queries on graph-modeled data originates in the '90s with the need of querying Web data, like HTML, which is heterogeneous and where data structure is not fully known.

A pioneering work in this field has been Lorel [1], a SQL/OQL-style query language designed to be implemented on top of an OODBMS. In later years, FleXPath [3] focused on this issue by dealing with XML data. Since then, data representation has been getting richer and richer of information, giving rise to a variety of graph-based data sources where edges between nodes not only express connection information, like in HTML/XML; rather edges can also carry semantic information on the relationships occurring between the connected data, like in RDF, thus empowering knowledge representation.

In this semantically richer context, the works which are more related to ours are [19, 10, 36, 24].

As far as we know, NAGA [19] is the first work which addresses the need of semantic query capabilities which go beyond keyword-based search as a key issue when searching for knowledge. Nevertheless, several differences exist between [19, 10], as well as [1, 3], and our work, specifically in query formulation and query answering. First of all, all these works do not allow for semantic approximations on query nodes' and edges labels. Then, edge approximation is expressed through the use of regular expressions over relationships as query edges' labels. Labels of matching paths must satisfy the given regular expression, thus following a pure syntactic approach. [10] also gives the possibility of specifying constraints on (sub)path lengths based on edge types in the regular expression. In very complex databases like those, for instance, in the biological field, finding data which exactly matches a complex regular expression may be a real challenge. On the other hand, simple regular expressions, i.e., made of a single label, are not approximated (except for the `isA` relationship in [19]).

A further difference is that GEX allows for specifying conditions to constrain query results through relational (indeed, also supported in [10]) and similarity predicates. Furthermore, in NAGA answers to relatedness queries, i.e., queries containing edges labeled by the special keyword `connect`, return nodes which are connected through any path in the data. As discussed in Section 3.1, topological connection does not imply that the data retrieved is meaningfully related. The same limitation is suffered by the other works. To overcome these limitations, our model relies on the Semantic Relatedness relation SR to exclude misleading results. Finally, NAGA does not delve into details as to the efficiency of the answering process, and only hints are given about the specific data structures and algorithms used to implement the system. On the other hand, in [10] a query graph is considered as a set of reachability queries, one for each query edge. This implies that a query result is a set of reachability query results, each one being a set of node pairs that are guaranteed to be connected through the specified regular expressions on the respective query edge. In GEX query

results are indeed graphs where connections between data nodes are completely specified.

The works in [36, 24] exploit schema information to derive the concepts of Meaningful Schema Pattern and Meaningful Query Focus, with similar objectives as our Semantic Relatedness relation. However, the proposed solution does not investigate the problem of handling labeled edges, and it is targeted for relational and XML data. Thus, it can be considered as an XML-like instantiation of SR.

Much research efforts have focused on querying graph databases. A large amount of work is devoted to finding exact matches of a query graph either in a given large graph [37, 39, 41] or in large sets of data graphs [33]. In this context, [35] introduces a limited kind of approximation by admitting query node label containment to deal with multi-labeled graphs, while [44] supports node label substring operations on RDF graphs. Some relevant works [14, 15, 17, 22] have the main goal of investigating efficiency issues, mostly focused on keyword-based search.

[29, 34, 8, 38, 40] go beyond the keyword-based query paradigm and support approximate subgraph matching. However, they only make syntactic considerations to evaluate the degree of structural approximations on query connections. In all these works the semantic relatedness of the connected data is not discussed. The Graph Information Discovery (GID) framework proposed in [31] founds on the notion of filters to output a ranked subgraph of an input graph. However, GID provides the user with limited query capabilities as to the specification of semantic relationships occurring between the data. In [42] query relaxations are applied to malleable schemas. Approximations are achieved by query expansion techniques based on the discovery of correlations of attributes and relationships in the data. However, query relaxations are not concerned with entities' labels, and thus the user must know the schema to start the query. Furthermore, approximations on relationships are limited to edge substitution, thus not considering structural relaxations to paths. A further relevant work is [7] which emphasizes the need of supporting flexible query answering over heterogeneous data sources. However, in that work the expressive power of queries is limited. For instance, our Query 2 in the reference example can not be expressed in the query language presented in [7]. Then, the work mainly focuses on indexing aspects, and approximations on queries are limited to the identification of synonyms. The proposal in [32] presents a hybrid query formalism which combines keyword search with structured queries, yet limited to tree-shaped unary queries, i.e., queries with only a single target variable. Furthermore, the work follows a pure IR approach where neither semantic nor structural approximations are considered for the query answering process.

As to query answering methodology, proposals like [1, 3, 42, 38] are orthogonal/dual to ours because they focus on generating query relaxations to be matched exactly in the data; the GEX query answering model instead founds on a *flexible query matching* mechanism that supports approximations both on the vocabulary and on the structure of a query. In line with this approach, GEX offers a ranking model that measures answer goodness and a top-k retrieval

algorithm that relies on it.

In the recent proposal [20] an innovative perspective is adopted: the match of a query graph may not necessarily be (even approximately) isomorphic to the query graph in terms of label and topologically equality. While opening an interesting issue, the work is limited to deal with graphs having unlabeled edges.

With regard to the efficiency of subgraph computation, an interesting approach is the one presented in [30] where a different perspective is taken: keywords, that are admitted to appear also as data graph edges' labels, are used to retrieve data subgraphs connecting them, which to derive structured queries from to be forwarded to a relational query engine. The main objective of this step is making query formulation more effective. Efficient algorithms are introduced for the computation of the top-k subgraphs. Similarly to our proposal, efficiency of subgraph exploration is obtained by means of a summary structure similar to a dataguide [13] where concept nodes are represented only once, in intensional form. The subgraph exploration algorithm that relies on it derives structured queries that are not guaranteed to return results, because the intensional representation possibly exposes data relationships that holds only for some data. Differently from [30] we exploit a summary structure to efficiently retrieve instances of concept nodes too. However, these are only those connected in a semantically meaningful way, being our structures founded on the SR relation. Further, our algorithms retrieve the top-k query results, while [30] retrieve the top-k queries. A further relevant work is [43] which, as said by its authors, is the first proposal that focuses on indexing both the structure and the labeling information of a large data graph to support approximate graph query answering. Differently from GEX, approximation is not semantic, in that it only admits the partial match of query nodes, while (unlabeled) query edges are allowed to be approximated by data paths.

As to the ranking of results, the models proposed in [8, 19] consist of a really valuable framework. However, these proposals are orthogonal to ours since they follow a statistical approach which exploits some knowledge of the underlying dataset. Our ranking model instead relies on scoring functions to evaluate the semantic approximations occurring at both data nodes and data edges. However, it would be interesting to incorporate such probabilistic principles into our model to enhance the ranking of results. The GID proposal [31] exploits authority-flow ranking techniques to support a query-customized ranking of results, i.e., it allows the user to specify what ranking mechanism (if any) should be used for each leg of the query. Nevertheless, since these techniques proved to be expensive when interactively applied, optimization techniques have to be necessarily employed. Similarly to our ranking model, the subgraph matching cost function presented in [20] is computed on the basis of the linear combination of label and neighborhood similarity for each query node. However such cost function is not designed to deal with partial query matches.

8. Conclusions

We presented the GEX (Graph-eXplorer) approach to support the approximate matching of complex queries on graph-modeled data.

GEX offers a framework that *generalizes* the existing approaches and allows for querying any datasets which conform to a graph representation. It exhibits a highly expressive graph-based query language for queries ranging from keyword-based to complex ones, by admitting several kinds of vague and missing information, according to varying degrees of knowledge the user may have.

GEX introduces the notion of Semantic Relatedness (SR) to identify meaningfully related nodes in a data graph. To the authors' knowledge, this is the first work that explores the employment of a semantic notion like SR for structural query approximation. The aim is that of allowing for meaningful structural relaxations only, by overcoming the limits of traditional topological-connection-based approaches.

The feasibility of the GEX framework is proved by the introduction of data structures and indices specifically designed to rely on the notion of SR. As an improvement of [25], the data structures have been completely redesigned, by definitely optimizing the space required to store the SR data. We have also shown that GEX is enhanced with a top-k retrieval algorithm that guarantees efficiency of execution, by exposing several cursor selection strategies, according to different policies for building the best ranked answers as soon as possible.

Finally, the extensive experimental evaluation on various real world datasets, in comparison with some relevant state-of-the-art proposals we simulated for the purpose, shows the effectiveness of GEX query answering as well as very good search time and indexing space performances. It is worth noting that the generality of GEX in dealing with different datasets does not affect the query answering mechanism proposed, rather it only impacts on the indices used to access the data.

In our future work, we plan to complete GEX with a model to support query formulation, by enabling the user to express queries in natural language.

Appendix A. An RDF-like Instantiation of SR

In this section, we show a possible instantiation of SR for an RDF-like data model, as introduced in Section 3.1.

Such an instantiation relies on the notion of type. More precisely, data edges are grouped together on the basis of the kind of relationship they represent. We assume the existence of five edge types: *property*, *type*, *isA*, *isPartOf*, and *domRel*. The type *property* is assigned to each edge between an instance node and a value node. A sample of *property* edge is the edge e_7 in the data graph in Fig. 1. *isA* is an acyclic transitive relation which expresses a hierarchical relationship between two classes while a *type* relation is used to link one entity node to a class it belongs to. All the edges in Fig. 1 labeled *subClassOf* and *type* are of type *isA* and *type*, respectively. *isPartOf* concerns the membership

of an instance to another instance and, finally, *domRel* denotes any relationship which can be established between instances. The data graph in Fig. 1 contains two *domRel* edges: e_8 and e_{10}. In the following, $\tau(e)$ will denote the type of the edge e.

The construction of SR on any data graph $\mathcal{G} = (N, E, L_N, L_E)$ conforming to an RDF-like data model is performed in an incremental fashion through a set of rewriting rules founded on the type semantics and the graph topology. The approach we adopted in introducing the above rules is conservative as it is exclusively based on types and does not make any assumption on the involved labels. On the other hand, adopting a "naïve" approach for the SR expansion could bring to misleading results. For instance, the author of a paper which cites another paper is not the author of the cited paper. Thus, we prefer not to overwhelm users with wrong results while accepting few false negatives. One way to overcome this problem is to adopt a fine grain semantic analysis of the chain of node and edge labels in order to state whether the starting node and the ending node are semantically related or not. This study is out of the scope of this paper and will be dealt with in our future work.

To this extent, similarly to [28] but for different purposes, the rewriting system starts from a set of axiomatic rules for edges in E, and recursively adds new node pairs by exploiting a set of extension rules. Each rule has a left-hand part which states preconditions and a right-hand part which specifies the properties of the node pair added to SR and the corresponding cost function c instance. The following axiomatic rule initializes SR with the set of edges in E[7]:

$$e \in E \quad \rightarrow \quad \{e|, \tau_{SR}(e) = \tau_{\mathcal{G}}(e), p(e) = e\}, c(e) = 0$$

while the following rules are used to associate each edge with a label:

$$\begin{aligned}
\tau_{\mathcal{G}}(e) = \mathit{isA} &\quad \rightarrow \quad \lambda_{SR}(e) = \texttt{isA} \\
\tau_{\mathcal{G}}(e) = \mathit{isPartOf} &\quad \rightarrow \quad \lambda_{SR}(e) = \texttt{isPartOf} \\
\tau_{\mathcal{G}}(e) = \mathit{type} &\quad \rightarrow \quad \lambda_{SR}(e) = \texttt{type} \\
\tau_{\mathcal{G}}(e) = \mathit{property} &\quad \rightarrow \quad \lambda_{SR}(e) = \lambda_{\mathcal{G}}(e) \\
\tau_{\mathcal{G}}(e) = \mathit{domRel} &\quad \rightarrow \quad \lambda_{SR}(e) = \lambda_{\mathcal{G}}(e)
\end{aligned}$$

Note that whenever the node semantics is carried by the type, the node is assigned a default label. Then, SR is extended by means of the rules shown in Tab. A.4.

In particular, in rule (r1) the path associated to the newly added edge is the concatenation of the two involved paths and the cost actually represents the difference between the length of $p(e'')$ and the length of a direct connection between x and z, i.e. 1.

For instance rule (1) applied to the data graph of Fig. 1 adds three edges to SR:

[7] We use subscripts $\mathcal{G}$ and SR to distinguish the properties in the graph and in the semantic relatedness relationship, respectively.

For all $e = (x, y), e' = (y, z) \in SR$:
(r1) $\tau(e) = \tau(e') = isA \text{OR} \tau(e) = type, \tau(e') = isA \text{OR } \tau(e) = \tau(e') = isPartOf \rightarrow$
$\{e'' = (x, z) | \tau(e'') = \tau(e), \lambda(e'') = \lambda(e), p(e'') = p(e) \circ {}^{8}p(e')\}, c(e'') = c(e) + c(e') + 1$

For all p_1, p_2 properties, $e = (x, y) \in SR$:
(r2) $\tau((p_1, p_2)) = isA, \lambda(e) = pd(p_1).\lambda \rightarrow \{e' = (x, y) | \tau(e') = pd(p_2).\tau, \lambda(e') = pd(p_2).\lambda, p(e') = p(e)\}$,
$c(e') = c(e) + 1$

For all p properties, $e = (x, y), e' = (y, z) \in SR$:
(r3) $\tau((p, \texttt{aTrans})) = type, \lambda(e) = pd(p).\lambda, \lambda(e') = pd(p).\lambda \rightarrow$
$\{e'' = (x, z) | \tau(e'') = pd(p).\tau, \lambda(e'') = pd(p).\lambda, p(e'') = p(e) \circ p(e')\}, c(e'') = c(e) + c(e') + 1$

For all p properties, cl classes, $e = (x, y), (p, cl) \in SR$:
(r4) $\tau((p, cl)) = domain, \lambda(e) = pd(p).\lambda \rightarrow \{e' = (x, cl) | \tau(e') = type, \lambda(e') = \texttt{type}, p(e') = p(e)\}$,
$c(e') = c(e) + 1$
(r5) $\tau((p, cl)) = range, \lambda(e) = pd(p).\lambda \rightarrow \{e' = (y, cl) | \tau(e') = type, \lambda(e') = \texttt{type}, p(e') = p(e)\}$,
$c(e') = c(e) + 1$

Table A.4: Rewriting rules for RDF-like graphs

e	$\tau(e)$	$\lambda(e)$	$p(e)$	$c(e)$
(n_6, n_1)	*type*	`type`	$\langle e_4, e_1 \rangle$	1
(n_7, n_1)	*type*	`type`	$\langle e_5, e_2 \rangle$	1
(n_{13}, n_1)	*type*	`type`	$\langle e_6, e_3 \rangle$	1

Then, we extend the RDF-like data model and the related rewriting system in order to take into account, besides the subclass hierarchies, other meaningful class properties which can be defined in OWL and RDFS. In particular, we focus on the *property* and *domRel* edges. For ease of reading, in the following we will refer to the *property* and *domRel* edges by using the generic term "property".

Properties in RDF-like graphs are defined by meta-level classes whose domains and ranges are defined through the newly added edge types *domain* and *range* and whose hierarchies are defined through *isA* edges. Moreover, whenever OWL descriptions are available, we consider the acyclic transitive property characteristic and allow this characteristic to be stated through the edge type *type* and the newly added class `aTrans`. For instance, the triples (`cite`, `type`, `aTrans`), (`cite`, `domain`, `Document`), and (`cite`, `range`, `Document`) could define property `cite`. Given this extended model, the rewriting system is extended with rules (r2)-(r5) above. Function $pd(\cdot)$ associates any meta-level class defining a property with the property it defines (the label $pd(\cdot).\lambda$ and the type $pd(\cdot).\tau$). To this end, we assume that the bound between each property defined at meta-level and its use at instance level is made through the involved labels which must be the same.

Rule (r2) states that all node pairs related by one property are also related by its superproperties, rule (r3) extends rule (r1) to any transitive property, while rule (r4) and (r5) exploit the property domain and range to add new class instances. Finally, it is worth noting that the cost associated to each edge added through rules (r2), (r4), and (r5) is not related to the path length but rather to the number of node pairs used to infer them.

Theorem 1 (convergence). *SR is finite and unique.*

Proof. First note that the rewriting system is monotone and finitely terminating. Moreover, it is also locally confluent. It follows that the rewriting system is globally confluent. Therefore SR is finite and unique. □

Finally, we did not discuss the two property characteristics, namely inverse and symmetric, due to the termination problem. Indeed, a repeated application of any rewriting rule exploiting such characteristics would add new instances to SR identical to some of the already included instances in SR, but paths and costs. Thus, the process would never terminate. However, as we adopt a distinct-node set semantics, we can directly compute $\overline{SR}$ instead of computing SR and, then, reducing it. In this case, we would add each "virtual" edge only once, the first time it is derived, as it can be shown that the cost of the instances added afterwards would be higher.

[1] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J.L. Wiener. The lorel query language for semistructured data. *Int. J. on Digital Libraries*, 1(1):68–88, 1997.

[2] G. Amato, F. Debole, P. Zezula, and F. Rabitti. YAPI: Yet Another Path Index for XML Searching. In *ECDL*, pages 176–187, 2003.

[3] S. Amer-Yahia, L.V.S. Lakshmanan, and S. Pandit. FleXPath: Flexible Structure and Full-Text Querying for XML. In *SIGMOD*, pages 83–94, 2004.

[4] R.A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[5] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword Searching and Browsing in Databases using BANKS. In *ICDE*, pages 431–440, 2002.

[6] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSEarch: A Semantic Search Engine for XML. In *VLDB*, pages 45–56, 2003.

[7] X. Dong and A.Y. Halevy. Indexing dataspaces. In *SIGMOD*, pages 43–54, 2007.

[8] S. Elbassuoni, M. Ramanath, R. Schenkel, M. Sydow, and G. Weikum. Language-model-based ranking for queries on rdf-graphs. In *CIKM*, pages 977–986, 2009.

[9] R. Fagin, A. Lotem, and M. Naor. Optimal Aggregation Algorithms for Middleware. In *PODS*, pages 102–113, 2001.

[10] W. Fan, J. Li, S. Ma, N. Tang, and Y. Wu. Adding Regular Expressions to Graph Reachability and Pattern Queries. In *ICDE*, pages 39–50, 2011.

[11] M.J. Franklin, A.Y. Halevy, and D. Maier. From databases to dataspaces: a new abstraction for information management. *SIGMOD Record*, 34(4):27–33, 2005.

[12] G.W. Furnas, T.K. Landauer, L.M. Gomez, and S.T. Dumais. The vocabulary problem in human-system communication. *Commun. ACM*, 30(11):964–971, 1987.

[13] R. Goldman and J. Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. In *VLDB*, pages 436–445, 1997.

[14] L. Guo, J. Shanmugasundaram, and G. Yona. Topology Search over Biological Databases. In *ICDE*, pages 556,565, 2007.

[15] H. He, H. Wang, J. Yang, and P.S. Yu. BLINKS: ranked keyword searches on graphs. In *SIGMOD*, pages 305–316, 2007.

[16] G.R. Hjaltason and H. Samet. Index-driven similarity search in metric spaces. *ACM Trans. Database Syst.*, 28(4):517–580, 2003.

[17] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional Expansion For Keyword Search on Graph Databases. In *VLDB*, pages 505–516, 2005.

[18] M. Kargar and A. An. Keyword Search in Graphs: Finding R-cliques. *Proc. VLDB Endow.*, 4(10):681–692, 2011.

[19] G. Kasneci, F.M. Suchanek, G. Ifrim, M. Ramanath, and G. Weikum. NAGA: Searching and Ranking Knowledge. In *ICDE*, pages 953–962, 2007.

[20] A. Khan, Y. Wu, C.C. Aggarwal, and X. Yan. NeMa: Fast Graph Search with Label Similarity. In *Proc. of the 39th Int. Conf. on Very Large Data Bases*, PVLDB'13, pages 181–192, 2013.

[21] C. Leacock and M. Chodorow. Combining Local Context and WordNet Similarity for Word Sense Identification. In C. Fellbaum, editor, *WordNet: An Electronic Lexical Database*, pages 256–283. MIT Press, 1998.

[22] G. Li, B.C. Ooi, J. Feng, J. Wang, and L. Zhou. EASE: An Effective 3-in-1 Keyword Search Method for Unstructured, Semi-structured and Structured Data. In *SIGMOD*, pages 903–914, 2008.

[23] Y. Li, C. Yu, and H.V. Jagadish. Schema-Free XQuery. In *VLDB*, pages 72–83, 2004.

[24] Y. Li, C. Yu, and H.V. Jagadish. Enabling Schema-Free XQuery with meaningful query focus. *The VLDB Journal*, 17(3):355–377, 2008.

[25] F. Mandreoli, R. Martoglia, W. Penzo, and G. Villani. Flexible Query Answering on Graph-modeled Data. In *EDBT*, pages 216–227, 2009.

[26] F. Mandreoli, R. Martoglia, and E. Ronchetti. Versatile Structural Disambiguation for Semantic-Aware Applications. In *CIKM*, pages 209–216, 2005.

[27] L. Qin, J.X. Yu, L. Chang, and Y. Tao. Querying communities in relational databases. In *ICDE*, pages 724–735, 2009.

[28] F.M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In *WWW*, pages 697–706, 2007.

[29] Y. Tian and J.M. Patel. TALE: A Tool for Approximate Large Graph Matching. In *ICDE*, pages 962–973, 2008.

[30] T. Tran, H. Wang, S. Rudolph, and P. Cimiano. Top-k Exploration of Query Candidates for Efficient Keyword Search on Graph-Shaped (RDF) Data. In *ICDE*, pages 405–416, 2009.

[31] R. Varadarajan, V. Hristidis, L. Raschid, M. Vidal, L. Ibáñez, and H. Rodríguez-Drumond. Flexible and efficient querying and ranking on hyperlinked data sources. In *EDBT*, pages 553–564, 2009.

[32] H. Wang, Q. Liu, T. Penin, L. Fu, L. Zhang, T. Tran, Y. Yu, and Y. Pan. Semplore: A scalable ir approach to search the web of data. *Journal of Web Semantics*, 7(3):177 – 188, 2009.

[33] Y. Xie and P.S. Yu. Cp-index: on the efficient indexing of large graphs. In *Proc. of the 20th ACM Int. Conf. on Information and Knowledge Management*, CIKM '11, pages 1795–1804, 2011.

[34] X. Yan, P.S. Yu, and J. Han. Substructure Similarity Search in Graph Databases. In *SIGMOD*, pages 766–777, 2005.

[35] J. Yang, S. Zhang, and W. Jin. Delta: indexing and querying multi-labeled graphs. In *Proc. of the 20th ACM Int. Conf. on Information and Knowledge Management*, CIKM '11, pages 1765–1774, 2011.

[36] C. Yu and H.V. Jagadish. Querying Complex Structured Databases. In *VLDB*, pages 1010–1021, 2007.

[37] S. Zhang, S. Li, and J. Yang. Gaddi: distance index based subgraph matching in biological networks. In *Proc. of the ACM 12th Int. Conf. on Extending Database Technology*, EDBT '09, pages 192–203, 2009.

[38] S. Zhang, J. Yang, and W. Jin. Sapper: subgraph indexing and approximate matching in large graphs. *Proc. VLDB Endow.*, 3(1-2):1185–1194, 2010.

[39] P. Zhao and J. Han. On graph query optimization in large networks. *Proc. VLDB Endow.*, 3(1-2):340–351, 2010.

[40] W. Zheng, L. Zou, X. Lian, D. Wang, and D. Zhao. Graph similarity search with edit distance constraint in large graph databases. In *Proc. of the 22nd ACM Int. Conf. on Information & Knowledge Management*, CIKM '13, pages 1595–1600, 2013.

[41] W. Zheng, L. Zou, X. Lian, H. Zhang, W. Wang, and D. Zhao. SQBC: An efficient subgraph matching method over large and dense graphs. *Information Sciences*, 261(0):116–131, 2014.

[42] X. Zhou, J. Gaugaz, W.-T. Balke, and W. Nejdl. Query relaxation using malleable schemas. In *SIGMOD*, pages 545–556, 2007.

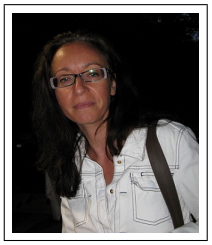
[43] L. Zhu, W.K. Ng, and J. Cheng. Structure and attribute index for approximate graph matching in large graphs. *Information Systems*, 36(6):958–972, 2011.

[44] L. Zou, J. Mo, L. Chen, M. Tamer Özsu, and D. Zhao. gStore: Answering SPARQL Queries via Subgraph Matching. *Proc. VLDB Endow.*, 4(8):482–493, 2011.

Author Biographies

Federica Mandreoli got the Laurea degree in Computer Science from the University of Bologna (Italy) in 1997. In 1997 she started her Ph.D. experience at DEIS (University of Bologna) and on March 2001 she received the Ph.D. degree in Computer Science Engineering with a thesis entitled "Temporal Schema Versioning in Object-Oriented Databases". She is currently an assistant professor in the FIM Department of the University of Modena and Reggio Emilia. Her research interests include information and knowledge management for large data sets, data sharing in P2P networks, and query processing over graph-structured data.

Riccardo Martoglia received his Laurea Degree (cum Laude) and his Ph.D. in Computer Engineering from the University of Modena and Reggio Emilia. He is currently an assistant professor in the FIM Department of the same university. His research themes are hot topics in the area of Databases, Information Systems, Information Retrieval and Semantic Web. In particular, his work concerns the study of new methodologies for managing, storing and querying large amounts of non-conventional information, including textual, XML and graph data. He is author of over 70 publications and has participated to many National and International projects on the above mentioned topics.



Wilma Penzo received her MS degree in Computer Science in 1993 from the University of Bologna, Italy, and her PhD degree in Electronic and Computer Engineering from the same University in 1997. Since 1996 she has been an Assistant Professor. She currently is with the Department of Computer Science and Engineering (DISI), University of Bologna. Her recent research interests include query processing on graph-based data, stream data management, Semantic Web, semantic P2P systems. She also dealt with fuzzy query languages for multimedia databases, semistructured databases, indexing and query processing in XML digital libraries.